ISO-ANSI Working Draft
Database Language SQL
(SQL3)

February 1993

ISO and ANSI SQL3 Working Draft—February 5, 1993

Digital Equipment Corporation
Maynard, Massachusetts

Contents

Page

Foreword xiii

Introduction xvii

1 Scope 1

2 Normative references 3

3 Definitions, notations, and conventions 7
3.1 Definitions 7
3.1.1 Definitions taken from ISO/IEC DIS 10646 7
3.1.2 Definitions taken from ISO 8601 7
3.1.3 Definitions provided in this Standard 7
3.2 Notation 11
3.3 Conventions 12
3.3.1 Informative elements 12
3.3.2 Specification of syntactic elements 12
3.3.3 Specification of the Information Schema 13
3.3.4 Use of terms 13
3.3.4.1 Exceptions 13
3.3.4.2 Syntactic containment 14
3.3.4.3 Terms denoting rule requirements 14
3.3.4.4 Rule evaluation order 15
3.3.4.5 Conditional rules 16
3.3.4.6 Syntactic substitution 16
3.3.4.7 Other terms 16
3.3.5 Descriptors 17
3.3.6 Index typography 17
3.4 Object identifier for Database Language SQL 18

4 Concepts 21
4.1 Data types 21
4.2 Character strings 22
4.2.1 Character strings and collating sequences 23
4.2.2 Operations involving character strings 24
4.2.2.1 Operators that operate on character strings and return character strings 24
4.2.2.2 Other operators involving character strings 24
4.2.3 Rules determining collating sequence usage 25
4.3 Bit strings 27
4.3.1 Bit string comparison and assignment 28
4.3.2 Operations involving bit strings 28
4.3.2.1 Operators that operate on bit strings and return bit strings 28
4.3.2.2 Other operators involving bit strings 28
4.4 Numbers 28
4.4.1 Characteristics of numbers 29

4.4.2 Operations involving numbers . . . 30
4.5 Enumerated types . . . 30
4.6 Boolean types . . . 30
4.7 Datetimes and intervals . . . 30
4.7.1 Datetimes . . . 31
4.7.2 Intervals . . . 32
4.7.3 Operations involving datetimes and intervals . . . 34
4.8 Object identifier type . . . 35
4.9 Abstract data types . . . 35
4.9.1 Encapsulation . . . 36
4.9.2 Subtypes and supertypes . . . 36
4.9.3 Type templates . . . 37
4.10 Collection types . . . 38
4.10.1 Set type . . . 38
4.10.2 Multiset type . . . 39
4.10.3 List type . . . 39
4.11 Type conversions and mixing of data types . . . 40
4.12 Domains . . . 41
4.13 Nulls . . . 41
4.14 Columns and attributes . . . 42
4.15 Tables, sets, and lists . . . 43
4.16 Tables . . . 44
4.17 Integrity constraints . . . 49
4.17.1 Checking of constraints . . . 49
4.17.2 Table constraints . . . 50
4.17.3 Domain constraints . . . 51
4.17.4 Assertions . . . 51
4.18 Functional dependencies and candidate keys . . . 51
4.18.1 General rules and definitions . . . 52
4.18.2 Functional dependencies in a base table . . . 53
4.18.3 Functional dependencies in <table value constructor> . . . 53
4.18.4 Functional dependencies in a <joined table> . . . 53
4.18.5 Functional dependencies in the result of a <from clause> . . . 54
4.18.6 Functional dependencies in the result of a <where clause> . . . 54
4.18.7 Functional dependencies in the result of a <group by clause> . . . 55
4.18.8 Functional dependencies in the result of a <having clause> . . . 55
4.18.9 Functional dependencies in a <query specification> . . . 55
4.18.10 Functional dependencies in a <query expression> . . . 56
4.18.11 Functional dependencies in a <recursive union> . . . 57
4.18.12 Selection of primary key or preferred candidate key . . . 57
4.19 Triggers . . . 57
4.19.1 Triggered actions . . . 58
4.19.2 Execution of triggered actions . . . 58
4.20 SQL-schemas . . . 58
4.21 Catalogs . . . 59

4.22 Clusters of catalogs . . . 60
4.23 SQL-data . . . 60
4.24 SQL-environment . . . 60
4.25 Modules . . . 61
4.26 Routines . . . 61
4.27 Parameters . . . 63
4.27.1 Status parameters . . . 64
4.27.2 Data parameters . . . 64
4.27.3 Indicator parameters . . . 65
4.28 Diagnostics area . . . 65
4.29 Standard programming languages . . . 65
4.30 Cursors . . . 66
4.31 SQL-statements . . . 68
4.31.1 Classes of SQL-statements . . . 68
4.31.2 SQL-statements classified by function . . . 69
4.31.3 Embeddable SQL-statements . . . 73
4.31.4 Preparable and immediately executable SQL-statements . . . 74
4.31.5 Directly executable SQL-statements . . . 76
4.31.6 SQL-statements and transaction states . . . 77
4.31.7 Asynchronous execution . . . 79
4.32 Embedded syntax . . . 79
4.33 SQL dynamic statements . . . 80
4.34 Direct invocation of SQL . . . 82
4.35 Privileges and roles . . . 82
4.36 SQL-agents . . . 84
4.37 SQL-transactions . . . 85
4.38 SQL-connections . . . 88
4.39 SQL-sessions . . . 89
4.40 Client-server operation . . . 91
4.41 Information Schema . . . 91
4.42 Leveling . . . 92
4.43 SQL Flagger . . . 93

5 Lexical elements . . . 95
5.1 <SQL terminal character> . . . 95
5.2 <token> and <separator> . . . 98
5.3 <literal> . . . 106
5.4 Names and identifiers . . . 115

6 Scalar expressions . . . 125
6.1 <data type> . . . 125
6.2 <value specification> and <target specification> . . . 133
6.3 <item reference> . . . 139
6.4 <attribute reference> . . . 141
6.5 <table reference> . . . 143
6.6 <column reference> and <object reference> . . . 146

6.7 <set function specification> . . . 149
6.8 <numeric value function> . . . 152
6.9 <string value function> . . . 155
6.10 <datetime value function> . . . 162
6.11 <reference conversion function> . . . 164
6.12 <case expression> . . . 165
6.13 <cast specification> . . . 168
6.14 <value expression> . . . 184
6.15 <numeric value expression> . . . 187
6.16 <string value expression> . . . 190
6.17 <enumerated value expression> . . . 194
6.18 <datetime value expression> . . . 196
6.19 <interval value expression> . . . 199
6.20 <table value expression> . . . 203
6.21 <boolean value expression> . . . 205

7 Query expressions . . . 207
7.1 <row value constructor> . . . 207
7.2 <table value constructor> . . . 210
7.3 <set value constructor> . . . 211
7.4 <multiset value constructor> . . . 213
7.5 <list value constructor> . . . 215
7.6 <table expression> . . . 217
7.7 <from clause> . . . 219
7.8 <joined table> . . . 221
7.9 <where clause> . . . 231
7.10 <group by clause> . . . 233
7.11 <having clause> . . . 235
7.12 <query specification> . . . 237
7.13 <query expression> . . . 247
7.14 <recursive union> . . . 260
7.15 <scalar subquery>, <row subquery>, and <table subquery> . . . 268

8 Predicates . . . 271
8.1 <predicate> . . . 271
8.2 <comparison predicate> . . . 273
8.3 <between predicate> . . . 278
8.4 <in predicate> . . . 280
8.5 <like predicate> . . . 282
8.6 <similar predicate> . . . 286
8.7 <null predicate> . . . 296
8.8 <quantified comparison predicate> . . . 298
8.9 <exists predicate> . . . 300
8.10 <unique predicate> . . . 301
8.11 <match predicate> . . . 302
8.12 <overlaps predicate> . . . 305

8.13 <quantified predicate> . . . 307
8.14 <there is predicate> . . . 309
8.15 <distinct predicate> . . . 310
8.16 <boolean predicate> . . . 313
8.17 <search condition> . . . 315

9 Data assignment rules and function determination . . . 317
9.1 Retrieval assignment . . . 317
9.2 Store assignment . . . 320
9.3 Set operation result data types and nullabilities . . . 323
9.4 Subject routine determination . . . 325

10 Additional common elements . . . 327
10.1 <interval qualifier> . . . 327
10.2 <language clause> . . . 331
10.3 <routine invocation> . . . 333
10.4 <generated type reference> . . . 341
10.5 <privileges> . . . 343
10.6 <character set specification> . . . 348
10.7 <collate clause> . . . 351
10.8 <constraint name definition> and <constraint attributes> . . . 352

11 Schema definition and manipulation . . . 355
11.1 <schema definition> . . . 355
11.2 <drop schema statement> . . . 358
11.3 <table definition> . . . 361
11.4 <column definition> . . . 372
11.5 <default clause> . . . 379
11.6 <table constraint definition> . . . 384
11.7 <unique constraint definition> . . . 386
11.8 <referential constraint definition> . . . 389
11.9 <check constraint definition> . . . 398
11.10 <alter table statement> . . . 401
11.11 <add column definition> . . . 403
11.12 <alter column definition> . . . 405
11.13 <set column default clause> . . . 406
11.14 <drop column default clause> . . . 407
11.15 <drop column domain clause> . . . 408
11.16 <drop column definition> . . . 410
11.17 <add supertable clause> . . . 413
11.18 <drop supertable clause> . . . 415
11.19 <add table constraint definition> . . . 416
11.20 <drop table constraint definition> . . . 417
11.21 <drop table statement> . . . 419
11.22 <view definition> . . . 421
11.23 <drop view statement> . . . 426

11.24 <domain definition> . . . 427
11.25 <alter domain statement> . . . 430
11.26 <set domain default clause> . . . 431
11.27 <drop domain default clause> . . . 432
11.28 <add domain constraint definition> . . . 433
11.29 <drop domain constraint definition> . . . 434
11.30 <drop domain statement> . . . 436
11.31 <null class definition> . . . 439
11.32 <drop null class statement> . . . 441
11.33 <character set definition> . . . 442
11.34 <drop character set statement> . . . 445
11.35 <collation definition> . . . 446
11.36 <drop collation statement> . . . 452
11.37 <translation definition> . . . 454
11.38 <drop translation statement> . . . 457
11.39 <assertion definition> . . . 459
11.40 <drop assertion statement> . . . 463
11.41 <trigger definition> . . . 464
11.42 <drop trigger statement> . . . 478
11.43 <drop routine statement> . . . 480
11.44 <abstract data type definition> . . . 483
11.45 <abstract data type body> . . . 485
11.46 <distinct type definition> . . . 493
11.47 <type template definition> . . . 495
11.48 <drop type template statement> . . . 497
11.49 <drop data type statement> . . . 498
11.50 <module definition> . . . 500
11.51 <drop module statement> . . . 502
11.52 <grant statement> . . . 503
11.53 <role definition> . . . 507
11.54 <grant role statement> . . . 508
11.55 <revoke role statement> . . . 510
11.56 <drop role statement> . . . 511
11.57 <revoke statement> . . . 512

12 Standard type templates and types . . . 521
12.1 Introduction . . . 521
12.2 SQL type templates and types . . . 521
12.2.1 Definitions used in this Subclause . . . 521
12.2.2 Concepts used in this Subclause . . . 521
12.2.3 SQL_Table type template . . . 522
12.2.4 SQL_Set type template . . . 526
12.2.5 SQL_List type template . . . 528
12.2.6 SQL_Empty_Table type . . . 531
12.2.7 SQL_Empty_Set type . . . 532
12.2.8 SQL_Empty_List type . . . 533

13 Module . . . 535
13.1 <module> . . . 535
13.2 <routine> . . . 539
13.3 <SQL procedure statement> . . . 553
13.4 Data type correspondences . . . 557
13.5 Rules for externally-invoked <routine>s . . . 570

14 Control statements . . . 581
14.1 <call statement> . . . 581
14.2 <return statement> . . . 582
14.3 <compound statement> . . . 584
14.4 <exception handler> . . . 587
14.5 <exception declaration> . . . 589
14.6 <SQL variable declaration> . . . 590
14.7 <assignment statement> . . . 592
14.8 <new statement> . . . 594
14.9 <destroy statement> . . . 596
14.10 <case statement> . . . 597
14.11 <if statement> . . . 599
14.12 <leave statement> . . . 601
14.13 <loop statement> . . . 602
14.14 <for statement . . . 604
14.15 <signal statement> . . . 606
14.16 <resignal statement> . . . 607

15 Data manipulation . . . 609
15.1 <declare cursor> . . . 609
15.2 <open statement> . . . 614
15.3 <fetch statement> . . . 617
15.4 <close statement> . . . 621
15.5 <select statement: single row> . . . 623
15.6 <delete statement: positioned> . . . 625
15.7 <delete statement: searched> . . . 628
15.8 <insert statement> . . . 631
15.9 <update statement: positioned> . . . 638
15.10 <update statement: searched> . . . 643
15.11 <temporary table declaration> . . . 649
15.12 <temporary view definition> . . . 652

16 Transaction management . . . 655
16.1 <set transaction statement> . . . 655
16.2 <set constraints mode statement> . . . 657
16.3 <test completion statement> . . . 659
16.4 <savepoint statement> . . . 661
16.5 <release savepoint statement> . . . 663
16.6 <commit statement> . . . 664

16.7 <rollback statement> . . . 666

17 Connection management . . . 669
17.1 <connect statement> . . . 669
17.2 <set connection statement> . . . 672
17.3 <disconnect statement> . . . 674

18 Session management . . . 677
18.1 <set catalog statement> . . . 677
18.2 <set schema statement> . . . 678
18.3 <set names statement> . . . 680
18.4 <set session authorization identifier statement> . . . 681
18.5 <set role statement> . . . 683
18.6 <set local time zone statement> . . . 684

19 Dynamic SQL . . . 687
19.1 Description of SQL item descriptor areas . . . 687
19.2 <allocate descriptor statement> . . . 692
19.3 <deallocate descriptor statement> . . . 694
19.4 <get descriptor statement> . . . 695
19.5 <set descriptor statement> . . . 699
19.6 <prepare statement> . . . 702
19.7 <deallocate prepared statement> . . . 708
19.8 <describe statement> . . . 709
19.9 <using clause> . . . 710
19.10 <execute statement> . . . 717
19.11 <execute immediate statement> . . . 719
19.12 <dynamic declare cursor> . . . 721
19.13 <allocate cursor statement> . . . 722
19.14 <dynamic open statement> . . . 724
19.15 <dynamic fetch statement> . . . 726
19.16 <dynamic close statement> . . . 728
19.17 <dynamic delete statement: positioned> . . . 729
19.18 <dynamic update statement: positioned> . . . 730
19.19 <preparable dynamic delete statement: positioned> . . . 732
19.20 <preparable dynamic update statement: positioned> . . . 733

20 Diagnostics management . . . 735
20.1 <get diagnostics statement> . . . 735

21 Embedded SQL . . . 747
21.1 <embedded SQL host program> . . . 747
21.2 <embedded exception declaration> . . . 755
21.3 <embedded SQL Ada program> . . . 760
21.4 <embedded SQL C program> . . . 763
21.5 <embedded SQL COBOL program> . . . 767
21.6 <embedded SQL Fortran program> . . . 771

21.7 <embedded SQL MUMPS program> . . . 774
21.8 <embedded SQL Pascal program> . . . 776
21.9 <embedded SQL PL/I program> . . . 779

22 Direct invocation of SQL . . . 783
22.1 <direct SQL statement> . . . 783
22.2 <direct select statement: multiple rows> . . . 787

23 Information Schema and Definition Schema . . . 789
23.1 Introduction . . . 789
23.2 Information Schema . . . 790
23.2.1 INFORMATION_SCHEMA Schema . . . 790
23.2.2 INFORMATION_SCHEMA_CATALOG_NAME base table . . . 791
23.2.3 INFORMATION_SCHEMA_CATALOG_NAME_CARDINALITY assertion . . . 792
23.2.4 SCHEMATA view . . . 793
23.2.5 DOMAINS view . . . 794
23.2.6 DOMAIN_CONSTRAINTS view . . . 796
23.2.7 TABLES view . . . 797
23.2.8 VIEWS view . . . 798
23.2.9 COLUMNS view . . . 799
23.2.10 ABSTRACT_DATA_TYPES view . . . 801
23.2.11 NULL_CLASSES view . . . 802
23.2.12 NULL_STATES view . . . 803
23.2.13 TABLE_PRIVILEGES view . . . 804
23.2.14 COLUMN_PRIVILEGES view . . . 805
23.2.15 USAGE_PRIVILEGES view . . . 806
23.2.16 ABSTRACT_DATA_TYPE_PRIVILEGES view . . . 807
23.2.17 ABSTRACT_DATA_TYPE_IMPACT_COLUMN view . . . 808
23.2.18 ABSTRACT_DATA_TYPE_IMPACT_DOMAIN view . . . 809
23.2.19 TABLE_CONSTRAINTS view . . . 810
23.2.20 REFERENTIAL_CONSTRAINTS view . . . 811
23.2.21 CHECK_CONSTRAINTS view . . . 812
23.2.22 KEY_COLUMN_USAGE view . . . 813
23.2.23 ASSERTIONS view . . . 814
23.2.24 CHARACTER_SETS view . . . 815
23.2.25 COLLATIONS view . . . 816
23.2.26 TRANSLATIONS view . . . 817
23.2.27 VIEW_TABLE_USAGE view . . . 818
23.2.28 VIEW_COLUMN_USAGE view . . . 819
23.2.29 CONSTRAINT_TABLE_USAGE view . . . 820
23.2.30 CONSTRAINT_COLUMN_USAGE view . . . 821
23.2.31 COLUMN_DOMAIN_USAGE view . . . 822
23.2.32 ROUTINES view . . . 823
23.2.33 TRIGGERS view . . . 824
23.2.34 TRIGGERED_ACTIONS view . . . 825
23.2.35 TRIGGERED_COLUMNS view . . . 826

23.2.36 SQL_LANGUAGES view . . . 827
23.2.37 SQL_IDENTIFIER domain . . . 828
23.2.38 CHARACTER_DATA domain . . . 828
23.2.39 CARDINAL_NUMBER domain . . . 829
23.3 Definition Schema . . . 830
23.3.1 Introduction . . . 830
23.3.2 DEFINITION_SCHEMA Schema . . . 831
23.3.3 USERS base table . . . 832
23.3.4 ROLES base table . . . 833
23.3.5 SCHEMATA base table . . . 834
23.3.6 ABSTRACT_DATA_TYPES base table . . . 835
23.3.7 DATA_TYPE_DESCRIPTOR base table . . . 836
23.3.8 DOMAINS base table . . . 841
23.3.9 DOMAIN_CONSTRAINTS base table . . . 843
23.3.10 TABLES base table . . . 845
23.3.11 VIEWS base table . . . 846
23.3.12 COLUMNS base table . . . 848
23.3.13 VIEW_TABLE_USAGE base table . . . 850
23.3.14 VIEW_COLUMN_USAGE base table . . . 851
23.3.15 TABLE_CONSTRAINTS base table . . . 852
23.3.16 KEY_COLUMN_USAGE base table . . . 854
23.3.17 REFERENTIAL_CONSTRAINTS base table . . . 856
23.3.18 CHECK_CONSTRAINTS base table . . . 858
23.3.19 CHECK_TABLE_USAGE base table . . . 859
23.3.20 CHECK_COLUMN_USAGE base table . . . 860
23.3.21 ABSTRACT_DATA_TYPE_PRIVILEGES base table . . . 861
23.3.22 ASSERTIONS base table . . . 862
23.3.23 PRIVILEGE_DESCRIPTORS base table . . . 863
23.3.24 ROLE_AUTHORIZATION_DESCRIPTORS base table . . . 865
23.3.25 NULL_CLASSES base table . . . 866
23.3.26 NULL_STATES base table . . . 867
23.3.27 TABLE_PRIVILEGES base table . . . 868
23.3.28 COLUMN_PRIVILEGES base table . . . 870
23.3.29 USAGE_PRIVILEGES base table . . . 872
23.3.30 CHARACTER_SETS base table . . . 874
23.3.31 COLLATIONS base table . . . 876
23.3.32 TRANSLATIONS base table . . . 878
23.3.33 ROUTINES base table . . . 880
23.3.34 TRIGGERS base table . . . 881
23.3.35 TRIGGERED_ACTIONS base table . . . 883
23.3.36 TRIGGERED_COLUMNS base table . . . 885
23.3.37 SQL_LANGUAGES base table . . . 886
23.3.38 USER_SESSION_ENABLED_ROLES base table . . . 889
23.3.39 Temporary views . . . 890
23.3.40 ENABLED_APPLICABLE_ROLES temporary view . . . 890

23.3.41 TABLE_PRIVILEGES temporary view . . . 891
23.3.42 COLUMN_PRIVILEGES temporary view . . . 892
23.3.43 ROLE_TABLE_GRANTS temporary view . . . 893
23.3.44 ROLE_COLUMN_GRANTS temporary view . . . 894
23.3.45 USER_ROLE_GRANTS temporary view . . . 895
23.3.46 ROLE_ROLE_GRANTS temporary view . . . 896
23.3.47 ENABLED_ROLES temporary view . . . 897
23.4 Assertions on the base tables . . . 898
23.4.1 UNIQUE_CONSTRAINT_NAME assertion . . . 898
23.4.2 EQUAL_KEY_DEGREES assertion . . . 899
23.4.3 KEY_DEGREE_GREATER_THAN_OR_EQUAL_TO_1 assertion . . . 900
23.4.4 SUB_TABLES base table . . . 901

24 Status codes . . . 903
24.1 SQLSTATE . . . 903
24.2 SQLCODE . . . 910

25 Conformance . . . 911
25.1 Introduction . . . 911
25.2 Claims of conformance . . . 911
25.3 Extensions and options . . . 912
25.4 Flagger requirements . . . 912
25.5 Processing methods . . . 913

Annex A Leveling the SQL Language . . . 915
A.1 Full SQL Specifications . . . 915
A.2 Intermediate SQL Specifications . . . 922
A.3 Entry SQL Specifications . . . 931

Annex B Implementation-defined elements . . . 943

Annex C Implementation-dependent elements . . . 955

Annex D Deprecated features . . . 963

Annex E Incompatibilities with X3.135-1992 and ISO/IEC 9075:1992 . . . 965

Annex F Maintenance and interpretation of SQL . . . 969

Index

TABLES

Table		Page
1	Collating coercibility rules for monadic operators	26
2	Collating coercibility rules for dyadic operators	26
3	Collating sequence usage for comparisons	26
4	Fields in datetime items	31
5	Fields in year-month INTERVAL items	33
6	Fields in day-time INTERVAL items	33
7	Valid values for fields in INTERVAL items	33
8	Valid operators involving datetimes and intervals	34
9	SQL-transaction isolation levels and the three phenomena	87
10	Valid values for fields in datetime items	130
11	Valid values for fields in INTERVAL items	131
12	Truth table for the AND boolean	206
13	Truth table for the OR boolean	206
14	<null predicate> semantics	296
15	Truth table for the IS boolean	313
16	Standard programming languages	332
17	Data type correspondences for Ada	557
18	Data type correspondences for C	559
19	Data type correspondences for COBOL	561
20	Data type correspondences for Fortran	564
21	Data type correspondences for MUMPS	566
22	Data type correspondences for Pascal	567
23	Data type correspondences for PL/I	569
24	Data types of <key word>s used in SQL item descriptor areas	689
25	Codes used for SQL data types in Dynamic SQL	690
26	Codes associated with datetime data types in Dynamic SQL	691
27	Codes used for <interval qualifier>s in Dynamic SQL	691
28	<identifier>s for use with <get diagnostics statement>	737
29	SQL-statement character codes for use in the diagnostics area	738
30	SQLSTATE class and subclass values	904
31	SQLCODE values	910

Foreword

ISO Only–SQL3

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at leat 75% of the national bodies casting a vote.

International Standard ISO/IEC 9075 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*.

This fourth edition cancels and replaces the third edition (ISO/iEC 9075:1992).

Annexes A, B, C, D, E, and F of this International Standard are for information only.

ANSI Only–SQL3

This Standard (American National Standard X3.135-199*x*, *Database Language—SQL*), is a revision of American National Standard X3.135-1992, (*Database Language*), that adds significant new features and capabilities to the specifications.

ANSI (the American National Standards Institute) is the United States national standards body charged with development of American National Standards.

This Standard was approved as an American National Standard by the American National Standards Institute on (**insert date here**).

Suggestions for improvement of this Standard are welcome. They should be sent to the Computer and Business Equipment Manufacturing Assocation, 311 First Street, NW, Washington, DC 20011.

This Standard was processed and approved for submittal to ANSI by the Accredited National Standards Committee on Information Processing Systems, X3. Committee approval of this Standard does not necessarily imply that all committee members voted for its approval. At the time that it approved this Standard, the X3 Committee had the following members.

(**Insert X3 officer and membership lists here.**)

Editor's Notes for X3H2-93-091 and YOK-003

American National Standard X3.135-199*x* was prepared by Technical Committee Group X3H2, Database Languages, working under the auspices of Accredited National Standards Committee X3, Information Processing Systems. Technical Committee X3H2 on Database, which developed this Standard, had the following members:

Donald R. Deutsch, Chair
Bruce M. Horowitz, Vice-Chair
Michael M. Gorman, Secretary
Leonard J. Gallagher, International Representative

(**Insert membership list here.**)

Others holding Technical Committee X3H2 membership while the committee was developing this Standard are the following:

(**Insert other membership list here.**)

This American Standard was approved in 199*x*.

This American Standard was developed from ANS X3.135-1992, Database Language SQL, and replaces that American Standard. It adds significant new features and capabilities to the specifications. It is generally compatible with ANS X3.135-1992 in the sense that, with very few exceptions, SQL language that conforms to ANS X3.135-1992 also conforms to this American Standard, and will be treated in the same way by an implementation of this American Standard as it would by an implementation of ANS X3.135-1992 The known incompatibilities between ANS X3.135-1992 and this American Standard are stated in Informative Annex E, "Incompatibilities with X3.135-1992 and ISO/IEC 9075:1992".

Technical changes between ANS X3.135-1992 and this American Standard include both improvements or enhancements to existing features and the definition of new features. Significant improvements in existing features include:

To Be Supplied.

Significant new features are:

1) Support for active ‘‘rules’’, called triggers.

2) Support for abstract data types.

3) Support for multiple null states.

4) Support for PENDANT referential integrity.

5) A recursive union operation for query expressions.

6) Support for enumerated and boolean data types.

7) Support for SENSITIVE cursors.

This American Standard contains six Informative annexes:

— Annex A (informative): Leveling the SQL Language;

— Annex B (informative): Implementation-defined elements;

— Annex C (informative): Implementation-dependent elements;

— Annex D (informative): Deprecated features;

— Annex E (informative): Incompatibilities with X3.135-1992 and ISO/IEC 9075:1992;

— Annex F (informative): Maintenance and interpretation of SQL.

Introduction

The organization of this
[ANSI] American
[ISO] International
Standard is as follows:

1) Clause 1, "Scope", specifies the scope of this
   [ANSI] American
   [ISO] International
   Standard.

2) Clause 2, "Normative references", identifies additional standards that, through reference in this
   [ANSI] American
   [ISO] International
   Standard, constitute provisions of this
   [ANSI] American
   [ISO] International
   Standard.

3) Clause 3, "Definitions, notations, and conventions", defines the notations and conventions used in this
   [ANSI] American
   [ISO] International
   Standard.

4) Clause 4, "Concepts", presents concepts used in the definition of SQL.

5) Clause 5, "Lexical elements", defines the lexical elements of the language.

6) Clause 6, "Scalar expressions", defines the elements of the language that produce scalar values.

7) Clause 7, "Query expressions", defines the elements of the language that produce rows and tables of data.

8) Clause 8, "Predicates", defines the predicates of the language.

9) Clause 9, "Data assignment rules and function determination", specifies the rules for assignments that retrieve data from or store data into the database, and formation rules for set operations.

10) Clause 10, "Additional common elements", defines additional language elements that are used in various parts of the language.

11) Clause 11, "Schema definition and manipulation", defines facilities for creating and managing a schema.

12) Clause 13, "Module", defines modules and procedures.

13) Clause 14, "Control statements", defines SQL-statements that control the execution of procedures.

14) Clause 15, "Data manipulation", defines the data manipulation statements.

15) Clause 16, "Transaction management", defines the SQL-transaction management statements.

16) Clause 17, "Connection management" defines the SQL-connection management statements.

17) Clause 18, "Session management", defines the SQL-session management statements.

18) Clause 19, "Dynamic SQL", defines the facilities for executing SQL-statements dynamically.

19) Clause 20, "Diagnostics management", defines the diagnostics management facilities.

20) Clause 21, "Embedded SQL", defines syntax for embedding SQL in certain standard programming languages.

21) Clause 22, "Direct invocation of SQL", defines the direct invocation of SQL language.

22) Clause 23, "Information Schema and Definition Schema", defines viewed tables that contain schema information.

23) Clause 24, "Status codes", defines values that identify the status of the execution of SQL-statements and the mechanisms by which those values are returned.

24) Clause 25, "Conformance", defines the criteria for conformance to this
ANSI American
ISO International
standard.

25) Annex A, "Leveling the SQL Language", is an informative Annex. It lists the leveling rules defining the Entry SQL and Intermediate SQL subset levels SQL levels of the SQL language.

26) Annex B, "Implementation-defined elements", is an informative Annex. It lists those features for which the body of the
ANSI American
ISO International
Standard states that the syntax or meaning or effect on the database is partly or wholly implementation-defined, and describes the defining information that an implementor shall provide in each case.

27) Annex C, "Implementation-dependent elements", is an informative Annex. It lists those features for which the body of the
ANSI American
ISO International
Standard states explicitly that the meaning or effect on the database is implementation-dependent.

28) Annex D, "Deprecated features", is an informative Annex. It lists features that the responsible Technical Committee intend will not appear in a future revised version of this
ANSI American
ISO International
Standard.

29) Annex E, "Incompatibilities with X3.135-1992 and ISO/IEC 9075:1992", is an informative Annex. It lists incompatibilities with the previous version of this
ANSI American

[ISO] International
Standard.

30) Annex F, "Maintenance and interpretation of SQL", is an informative Annex. It identifies SQL interpretations and corrections that have been processed by
[ANSI] ANSI Accredited Committee X3
[ISO] ISO/IEC JTC1/SC21
since adoption of
[ANSI] ANSI X3.135-1989.
[ISO] ISO/IEC 9075:1989.

In the text of this
[ANSI] American
[ISO] International
Standard, Clauses begin a new odd-numbered page, and in Clause 5, "Lexical elements", through Clause 24, "Status codes", Subclauses begin a new page. Any resulting blank space is not significant.

Information Technology — Database Languages — SQL

1 Scope

This
ANSI American
ISO International
Standard defines the data structures and basic operations on SQL-data. It provides functional capabilities for creating, accessing, maintaining, controlling, and protecting SQL-data.

Note: The framework for this
ANSI American
ISO International
Standard is described by the Reference Model of Data Management (ISO DIS 10032:1991).

This
ANSI American
ISO International
Standard specifies the syntax and semantics of a database language:

— for specifying and modifying the structure and the integrity constraints of SQL-data;

— for declaring and invoking operations on SQL-data and cursors; and

— for declaring database language procedures and embedding them into a standard programming language.

It also specifies an Information Schema that describes the structure and the integrity constraints of SQL-data.

This
ANSI American
ISO International
Standard provides a vehicle for portability of data definitions and compilation units between SQL-implementations.

This
ANSI American
ISO International
Standard provides a vehicle for interconnection of SQL-implementations.

This
ANSI American
ISO International
Standard specifies syntax for embedding SQL-statements in a compilation unit that otherwise conforms to the standard for a particular programming language. It defines how an equivalent compilation unit may be derived that conforms to the particular programming language standard. In that equivalent compilation unit, each embedded SQL-statement has been replaced by statements that invoke a database language procedure that contains the SQL-statement.

This
ANSI American
ISO International
Standard specifies syntax for direct invocation of SQL-statements.

This
ANSI American
ISO International
Standard does not define the method or time of binding between any of:

— database management system components,

— SQL data definition declarations,

— SQL procedures, or

— compilation units, including those containing embedded SQL.

Implementations of this
ANSI American
ISO International
Standard may exist in environments that also support application programming languages, end-user query languages, report generator systems, data dictionary systems, program library systems, and distributed communication systems, as well as various tools for database design, data administration, and performance optimization.

2 Normative references

The following standards contain provisions that, through reference in this text, constitute provisions of this
ANSI American
ISO International
Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this
ANSI American
ISO International
Standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ANSI Only–SQL3

— ANSI X3.9-1978, *American National Standard Programming Language FORTRAN.*

— ANSI X3.23-1985, *American National Standard for Information Systems—Programming Language—COBOL.*

— ANSI X3.53-1976, *American National Standard Programming Language PL/I.*

— ANSI X3.159-1989, *American National Standard for Information Systems—Programming Language—C.*

— ANSI X3.198-1991, *American National Standard for Information Systems—Programming Language—Fortran.*

Note: ANSI X3.198-1991 introduces no incompatibilities with ANSI X3.9-1978 that affect the binding between Fortran and SQL; therefore, wherever ‘‘Fortran’’ is specified in this
ANSI American
ISO International
Standard, ANSI X3.198-1991 is implicit.

— ANSI/MDC X11.1-1990, *American National Standard for Information Systems—Programming Language—MUMPS.*

— ANSI/IEEE 770/X3.97-1983, *American National Standard for Information Systems—Programming Language—Pascal.*

— ANSI/IEEE 770/X3.160-1989, *American National Standard for Information Systems—Programming Language—Extended Pascal.*

— ANSI/MIL-STD-1815A-1983, *American National Standard for Information Systems—Reference Manual for the Ada® Programming Language*.

— ISO/IEC 646:1991, *Information technology—ISO 7-bit coded character set for information interchange*.

— ISO 2022:1986, *Information technology—ISO 7-bit and 8-bit coded character sets—code extension techniques*.

— ISO 8601:1988, *Data elements and interchange formats – Information interchange—Representation of dates and times*.

— ISO/IEC 8824:1990, *Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1)*.

— ISO/IEC 9579-2:[1], *Information technology – Open Systems Interconnection — Remote Database Access, Part 2: SQL specialization*.

— ISO/IEC 10646:[1], *Information technology – Multiple-octet coded character set*.

— ISO/IEC CD 11404, *Common Language-Independent Datatypes (CLID)*.

ISO Only–SQL3

— ISO/IEC 646:1991, *Information technology—ISO 7-bit coded character set for information interchange*.

— ISO/IEC 1539:1991, *Information technology—Programming languages—Fortran.*

— ISO 1989:1985, *Programming languages—COBOL.*
(Endorsement of ANSI X3.23-1985).

— ISO 2022:1986, *Information technology—ISO 7-bit and 8-bit coded character sets—code extension techniques*.

— ISO 6160:1979, *Programming languages—PL/I*
(Endorsement of ANSI X3.53-1976).

— ISO 7185:1990, *Information technology—Programming languages—Pascal*.

— ISO 8601:1988, *Data elements and interchange formats – Information interchange—Representation of dates and times*.

— ISO 8652:1987, *Programming languages—Ada*.
(Endorsement of ANSI/MIL-STD-1815A-1983).

— ISO/IEC 8824:1990, *Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1)*.

— ISO/IEC 9579-2:[1], *Information technology – Open Systems Interconnection — Remote Database Access, Part 2: SQL specialization*.

— ISO/IEC 9899:1990, *Programming languages – C*.

— ISO/IEC 10206:1991, *Information technology—Programming languages—Extended Pascal.*

— ISO/IEC 10646:[1], *Information technology – Multiple-octet coded character set*.

— ISO/IEC CD 11404, *Common Language-Independent Datatypes (CLID)*.

— ISO/IEC 11756:[1], *Information technology—Programming languages—MUMPS*.

[1] To be published

3 Definitions, notations, and conventions

3.1 Definitions

For the purposes of this
ANSI American
ISO International
Standard, the following definitions apply.

3.1.1 Definitions taken from ISO/IEC DIS 10646

This
ANSI American
ISO International
Standard makes use of the following terms defined in ISO/IEC 10646:

a) **character**

b) **octet**

c) **variable-length coding**

d) **fixed-length coding**

3.1.2 Definitions taken from ISO 8601

This
ANSI American
ISO International
Standard makes use of the following terms defined in ISO 8601:

a) **Coordinated Universal Time** (UTC)

b) **date** (‘‘date, calendar’’ in ISO 8601)

3.1.3 Definitions provided in this Standard

This
ANSI American
ISO International
Standard defines the following terms:

a) **abstract data type—ADT**: Specification of behavior of similar objects, called *instances* of that ADT. An important aspect of ADTs is the separation of the *interface* of the type from its *implementation*.

b) **actor operations**: Operations that are either observer operations or mutator operations.

c) **assignable**: The characteristic of a value or of a data type that permits that value or the values of that data type to be assigned to data instances of a specified data type. See Subclause 4.11, "Type conversions and mixing of data types".

d) **attribute**: An attribute may be stored or virtual. A stored attribute is a component (of an ADT) that has a name and a data type and whose name can be referenced where permitted by the encapsulation rules. A virtual attribute does not have a stored value but simulates the behavior of a stored atrribute by means of a *get function* to get its value and, optionally, a *set function* to assign to, or update, its value.

e) **cardinality** (of a collection): The number of objects in that collection. Those objects need not necessarily have distinct values.

f) **character repertoire**; **repertoire**: A set of characters used for a specific purpose or application. Each character repertoire has an implied default collating sequence.

g) **coercibility**: An attribute of character string data items that governs how a collating sequence for the items is determined.

h) **collation**; **collating sequence**: A method of ordering two comparable character strings. Every character set has a default collation.

i) **comparable**: The characteristic of two data objects that permits the value of one object to be compared with the value of the other object. Also said of data types: Two data types are comparable if objects of those data types are comparable. If one of the two data types is an abstract data type, then both must be the same abstract data type.

j) **component (of an ADT)**: One element of the representation.

k) **constructor operations**: Operations that create new instances of an ADT as SQL-data.

l) **destructor operations**: Operations that remove SQL-data that represents an instance of an ADT.

m) **descriptor**: A coded description of an SQL object. It includes all of the information about the object that a conforming SQL-implementation requires.

n) **distinct**: Two values are said to be not distinct if either: both are the null value, or they compare equal according to Subclause 8.2, "<comparison predicate>". Otherwise they are distinct. Two rows (or partial rows) are distinct if at least one of their pairs of respective values is distinct. Otherwise they are not distinct. The result of evaluating whether or not two values or two rows are distinct is never unknown.

o) **duplicate**: Two or more values or rows are said to be duplicates (of each other) if and only if they are not distinct.

p) **dyadic operator**: An operator having two operands: a left operand and a right operand. An example of a dyadic arithmetic operator in this
ANSI American
ISO International
Standard is '' −'', specifying the subtraction of the right operand from the left operand.

q) **encapsulation**: A mechanism to limit visibility of certain members of an ADT. Each member of an ADT is specified with an encapsulation level of either public, private, or protected. A private member is visible only within the definition of the ADT itself (including its operations), whereas a public member is also visible outside the ADT definition wherever the ADT itself is visible. A

protected member is intermediate between these, being visible only within the definition of the ADT and the definition of any subtype of the ADT.

r) **form-of-use**: A convention (or encoding) for representing characters (in character strings). Some forms-of-use are fixed-length codings and others are variable-length codings.

s) **form-of-use conversion**: A method of converting character strings from one form-of-use to another form-of-use.

t) **generated type**: an ADT resulting from specifying the name of a type template and a set of template parameters required by that type template

u) **implementation (of an ADT)**: Some of the operations associated with an ADT might be realized by means of data that is stored as SQL-data, while other operations might be realized as executable code (functions) that implements the manipulations that can be performed on, to, or with, an object. Stored data together with the data structures and code that implement the behavior of an ADT is its *implementation*.

v) **implementation-defined**: Possibly differing between SQL-implementations, but specified by the implementor for each particular SQL-implementation.

w) **implementation-dependent**: Possibly differing between SQL-implementations, but not specified by this
ANSI American
ISO International
Standard and not required to be specified by the implementor for any particular SQL-implementations.

x) **interface (of an ADT)**: The behavior of an ADT, specified through a set of those attributes and operation signatures that are not encapsulated.

y) **member**: An attribute or operation of an ADT.

z) **monadic operator**: An operator having one operand. An example of a monadic arithmetic operator in this
ANSI American
ISO International
Standard is −, specifying the negation of the operand.

aa) **multiset**: An unordered collection of objects that are not necessarily distinct. The collection may be empty.

bb) **mutator operations**: Operations that change some part of the state of an ADT instance, given the ADT instance and a new value.

cc) ***n*-adic operator**: An operator having a variable number of operands (informally: *n* operands). An example of an *n*-adic operator in this
ANSI American
ISO International
Standard is COALESCE.

dd) **null value (null)**: A special value, or mark, that is used to indicate the absence of any data value.

ee) **object identifier (OID)**: An immutable unique value associated with an ADT instance that is independent of the ADT value.

ff) **observer operations**: Operations that do not change the state of an ADT instance and may return values derived from the value of an ADT instance, given the ADT instance.

gg) **persistent**: Continuing to exist indefinitely, until destroyed deliberately. Referential and cascaded actions are regarded as deliberate. Actions incidental to the ending of an SQL-transaction (see Subclause 4.37, "SQL-transactions") or an SQL-session (see Subclause 4.39, "SQL-sessions") are not regarded as deliberate.

hh) **redundant duplicates**: All except one of any multiset of duplicate values or rows.

ii) **repertoire**: See **character repertoire**.

jj) **representation (of an ADT)**: The ordered sequence of stored attributes of the ADT, including both those that are encapsulated and those that are not.

kk) **SQL-implementation**: A database management system that conforms to this
ANSI American
ISO International
Standard.

ll) **sequence**: An ordered collection of objects that are not necessarily distinct.

mm) **set**: An unordered collection of distinct objects. The collection may be empty.

ANSI Only–SQL3

---

nn) **signature (of a routine)**: the name of a routine, the position and data types of each of its parameters, and an indication of whether it is a function or a procedure.

ISO Only—caused by ANSI changes not yet considered by ISO

---

oo) **signature (of an ADT interface operation)**: The name of an operation and the names, positions, and data types of each of its arguments.

---

pp) **state (of an ADT instance)**: The ordered sequence of stored components of an ADT instance, excluding any component named OID.

qq) **template parameters**: A type template is defined with one or more formal template parameters. A generated type is denoted by a <generated type reference>, which consists of the name of a type template and a set of actual template parameters. Template parameters may be value template parameters or type template parameters.

rr) **translation**: A method of translating characters in one character repertoire into characters of the same or a different character repertoire.

ss) **type template**: the definition of a ‘‘meta type’’ that is used to specify a generated type.

tt) **type template parameter**: a template parameter used to specify a <data type>.

uu) **value (of an ADT instance)**: See *state*.

vv) **value template parameter**: a template parameter used to specify a value of some <data type>.

3.2 Notation

The syntactic notation used in this
ANSI American
ISO International
Standard is an extended version of BNF (''Backus Normal Form'' or ''Backus Naur Form'').

In BNF, each syntactic element of the language is defined by means of a *production rule*. This defines the element in terms of a formula consisting of the characters, character strings, and syntactic elements that can be used to form an instance of it.

The version of BNF used in this
ANSI American
ISO International
Standard makes use of the following symbols:

Symbol	*Meaning*
< >	Angle brackets delimit character strings that are the names of syntactic elements, the non-terminal symbols of the SQL language.
::=	The definition operator. This is used in a production rule to separate the element defined by the rule from its definition. The element being defined appears to the left of the operator and the formula that defines the element appears to the right.
[ ]	Square brackets indicate optional elements in a formula. The portion of the formula within the brackets may be explicitly specified or may be omitted.
{ }	Braces group elements in a formula. The portion of the formula within the braces must be explicitly specified.
|	The alternative operator. The vertical bar indicates that the portion of the formula following the bar is an alternative to the portion preceding the bar. If the vertical bar appears at a position where it is not enclosed in braces or square brackets, it specifies a complete alternative for the element defined by the production rule. If the vertical bar appears in a portion of a formula enclosed in braces or square brackets, it specifies alternatives for the contents of the innermost pair of such braces or brackets.
. . .	The ellipsis indicates that the element to which it applies in a formula may be repeated any number of times. If the ellipsis appears immediately after a closing brace ''}'', then it applies to the portion of the formula enclosed between that closing brace and the corresponding opening brace ''{''. If an ellipsis appears after any other element, then it applies only to that element.
!!	Introduces normal English text. This is used when the definition of a syntactic element is not expressed in BNF.

Spaces are used to separate syntactic elements. Multiple spaces and line breaks are treated as a single space. Apart from those symbols to which special functions were given above, other characters and character strings in a formula stand for themselves. In addition, if the symbols to the right of the definition operator in a production consist entirely of BNF symbols, then those symbols stand for themselves and do not take on their special meaning.

Pairs of braces and square brackets may be nested to any depth, and the alternative operator may appear at any depth within such a nest.

A character string that forms an instance of any syntactic element may be generated from the BNF definition of that syntactic element by application of the following steps:

1) Select any one option from those defined in the right hand side of a production rule for the element, and replace the element with this option.

2) Replace each ellipsis and the object to which it applies with one or more instances of that object.

3) For every portion of the string enclosed in square brackets, either delete the brackets and their contents or change the brackets to braces.

4) For every portion of the string enclosed in braces, apply steps 1 through 5 to the substring between the braces, then remove the braces.

5) Apply steps 1 through 5 to any non-terminal syntactic element (i.e., name enclosed in angle brackets) that remains in the string.

The expansion or production is complete when no further non-terminal symbols remain in the character string.

3.3 Conventions

3.3.1 Informative elements

In several places in the body of this
ANSI American
ISO International
Standard, informative notes appear. For example:

Note: This is an example of a note.

Those notes do not belong to the normative part of this
ANSI American
ISO International
Standard and conformance to material specified in those notes shall not be claimed.

3.3.2 Specification of syntactic elements

Syntactic elements are specified in terms of:

— **Function**: A short statement of the purpose of the element.

— **Format**: A BNF definition of the syntax of the element.

— **Syntax Rules**: A specification of the syntactic properties of the element, or of additional syntactic constraints, not expressed in BNF, that the element shall satisfy, or both.

— **Access Rules**: A specification of rules governing the accessibility of schema objects that must hold before the General Rules may be successfully applied.

****Editor's Note****
The precise algorithm for determining *what* an Access Rule is has not been well-specified. See Possible Problems 013 and 029 in the Editor's Notes.

— **General Rules**: A specification of the run-time effect of the element. Where more than one General Rule is used to specify the effect of an element, the required effect is that which would be obtained by beginning with the first General Rule and applying the Rules in numerical sequence unless a Rule is applied that specifies or implies a change in sequence or termination of the application of the Rules. Unless otherwise specified or implied by a specific Rule that is applied, application of General Rules terminates when the last in the sequence has been applied.

— **Leveling Rules**: A specification of how the element must be supported for each of the levels of SQL.

The scope of notational symbols is the Subclause in which those symbols are defined. Within a Subclause, the symbols defined in Syntax Rules, Access Rules, or General Rules can be referenced in other rules provided that they are defined before being referenced.

3.3.3 Specification of the Information Schema

The objects of the Information Schema in this
ANSI American
ISO International
Standard are specified in terms of:

— **Function**: A short statement of the purpose of the definition.

— **Definition**: A definition, in SQL, of the object being defined.

— **Description**: A specification of the run-time value of the object, to the extent that this is not clear from the definition.

The definitions used to define the views in the Information Schema are used only to specify clearly the contents of those viewed tables. The actual objects on which these views are based are implementation-dependent.

3.3.4 Use of terms

3.3.4.1 Exceptions

The phrase ‘‘an exception condition is raised:’’, followed by the name of a condition, is used in General Rules and elsewhere to indicate that the execution of a statement is unsuccessful, application of General Rules. other than those of Subclause 13.2, "<routine>", and Subclause 22.1, "<direct SQL statement>", may be terminated, diagnostic information is to be made available, and that execution of the statement is to have no effect on SQL-data or schemas. The effect on <target specification>s and SQL descriptor areas of an SQL-statement that terminates with an exception condition, unless explicitly defined by this
ANSI American
ISO International
standard, is implementation-dependent.

The phrase ‘‘a completion condition is raised:’’, followed by the name of a condition, is used in General Rules and elsewhere to indicate that application of General Rules is not terminated and diagnostic information is to be made available; unless an exception condition is also raised, the execution of the statement is successful.

If more than one condition could have occurred as a result of a statement, it is implementation-dependent whether diagnostic information pertaining to more than one condition is made available.

3.3.4.2 Syntactic containment

In a Format, a syntactic element <A> is said to *immediately contain* a syntactic element <B> if <B> appears on the right-hand side of the BNF production rule for <A>. A syntactic element <A> is said to *contain* or *specify* a syntactic element <C> if <A> immediately contains <C> or if <A> immediately contains a syntactic element <B> that contains <C>.

In SQL language, an instance *A1* of <A> is said to *immediately contain* an instance *B1* of <B> if <A> immediately contains <B> and the text of *B1* is part of the text of *A1*. An instance *A1* of <A> is said to *contain* or *specify* an instance *C1* of <C> if *A1* immediately contains *C1* or if *A1* immediately contains an instance *B1* of <B> that contains *C1*.

An instance *A1* of <A> is said to contain an instance *B1* of <B> *with an intervening* <C> if *A1* contains *B1* and *A1* contains an instance *C1* of <C> that contains *B1*. An instance *A1* of <A> is said to contain an instance *B1* of <B> *without an intervening* <C> if *A1* contains *B1* and *A1* does not contain an instance *C1* of <C> that contains *B1*.

An instance *A1* of <A> *simply contains* an instance *B1* of <B> if *A1* contains *B1* without an intervening instance *A2* of <A> or an intervening instance *B2* of <B>.

If <A> contains <B>, then <B> is said to be *contained in* <A> and <A> is said to be a *containing* production symbol for <B>. If <A> simply contains <B>, then <B> is said to be *simply contained in* <A> and <A> is said to be a *simply containing* production symbol for <B>.

Let *A1* be an instance of <A> and let *B1* be an instance of <B>. If <A> contains <B>, then *A1* is said to *contain B1* and *B1* is said to be *contained in A1*. If <A> simply contains <B>, then *A1* is said to *simply contain B1* and *B1* is said to be *simply contained in A1*.

An instance *A1* of <A> is the *innermost* <A> satisfying a condition *C* if *A1* satisfies *C* and *A1* does not contain an instance *A2* of <A> that satisfies *C*. An instance *A1* of <A> is the *outermost* <A> satisfying a condition *C* if *A1* satisfies *C* and *A1* is not contained in an instance *A2* of <A> that satisfies *C*.

If <A> contains a <table name> that identifies a view that is defined by a <view definition> *V*, then <A> is said to *generally contain* the <query expression> contained in *V*. If <A> contains <B>, then <A> generally contains <B>. If <A> generally contains <B> and <B> generally contains <C>, then <A> generally contains <C>.

An instance *A1* of <A> *directly contains* an instance *B1* of <B> if *A1* contains *B1* without an intervening <set function specification> or <subquery>.

3.3.4.3 Terms denoting rule requirements

In the Syntax Rules, the term *shall* defines conditions that are required to be true of syntactically conforming SQL language. When such conditions depend on the contents of the schema, then they are required to be true just before the actions specified by the General Rules are performed. The treatment of language that does not conform to the SQL Formats and Syntax Rules is implementation-dependent. If any condition required by Syntax Rules is not satisfied when the evaluation of Access or General Rules is attempted and the implementation is neither processing non-conforming SQL language nor processing conforming SQL language in a non-conforming manner, then an exception condition is raised: *syntax error or access rule violation* (if this situation occurs during dynamic execution of an SQL-statement, then the exception that is raised is *syntax*

error or access rule violation in dynamic SQL statement; if the situation occurs during direct invocation of an SQL-statement, then the exception that is raised is *syntax error or access rule violation in direct SQL statement*).

In the Access Rules, the term *shall* defines conditions that are required to be satisfied for the successful application of the General Rules. If any such condition is not satisfied when the General Rules are applied, then an exception condition is raised: *syntax error or access rule violation* (if this situation occurs during dynamic execution of an SQL-statement, then the exception that is raised is *syntax error or access rule violation in dynamic SQL statement*; if the situation occurs during direct invocation of an SQL-statement, then the exception that is raised is *syntax error or access rule violation in direct SQL statement*).

In the Leveling Rules, the term *shall* defines conditions that are required to be true of SQL language for it to syntactically conform to the specified level of conformance.

3.3.4.4 Rule evaluation order

A conforming implementation is not required to perform the exact sequence of actions defined in the General Rules, but shall achieve the same effect on SQL-data and schemas as that sequence. The term *effectively* is used to emphasize actions whose effect might be achieved in other ways by an implementation.

The Syntax Rules and Access Rules for contained syntactic elements are effectively applied at the same time as the Syntax Rules and Access Rules for the containing syntactic elements. The General Rules for contained syntactic elements are effectively applied before the General Rules for the containing syntactic elements. Where the precedence of operators is determined by the Formats of this
ANSI American
ISO International
Standard or by parentheses, those operators are effectively applied in the order specified by that precedence. Where the precedence is not determined by the Formats or by parentheses, effective evaluation of expressions is *generally* performed from left to right. However, it is implementation-dependent whether expressions are *actually* evaluated left to right, particularly when operands or operators might cause conditions to be raised or if the results of the expressions can be determined without completely evaluating all parts of the expression. In general, if some syntactic element contains more than one other syntactic element, then the General Rules for contained elements that appear earlier in the production for the containing syntactic element are applied before the General Rules for contained elements that appear later.

For example, in the production:

<A> ::= <B> <C>

the Syntax Rules and Access Rules for <A>, <B>, and <C> are effectively applied simultaneously. The General Rules for <B> are applied before the General Rules for <C>, and the General Rules for <A> are applied after the General Rules for both <B> and <C>.

If the result of an expression or search condition can be determined without completely evaluating all parts of the expression or search condition, then the parts of the expression or search condition whose evaluation is not necessary are called the *inessential* parts. If the Access Rules pertaining to inessential parts are not satisfied, then the *syntax error or access rule violation* exception condition is raised regardless of whether or not the inessential parts are actually evaluated. If evaluation of the inessential parts would cause an exception condition to be raised, then it is implementation-dependent whether or not that exception condition is raised.

3.3.4.5 Conditional rules

Conditional rules are specified with ''If'' or ''Case'' conventions. Rules specified with ''Case'' conventions include a list of conditional sub-rules using ''If'' conventions. The first such ''If'' sub-rule whose condition is true is the effective sub-rule of the ''Case'' rule. The last sub-rule of a ''Case'' rule may specify ''Otherwise''. Such a sub-rule is the effective sub-rule of the ''Case'' rule if no preceding ''If'' sub-rule in the ''Case'' rule has a true condition.

3.3.4.6 Syntactic substitution

In the Syntax and General Rules, the phrase ''*X* is implicit'' indicates that the Syntax and General Rules are to be interpreted as if the element *X* had actually been specified.

In the Syntax and General Rules, the phrase ''the following <*X*> is implicit: *Y*'' indicates that the Syntax and General Rules are to be interpreted as if a syntactic element <*X*> containing *Y* had actually been specified.

In the Syntax Rules and General Rules, the phrase ''*former* is equivalent to *latter*'' indicates that the Syntax Rules and General Rules are to be interpreted as if all instances of *former* in the element had been instances of *latter*.

If a BNF nonterminal is referenced in a Subclause without specifying how it is contained in a BNF production that the Subclause defines, then

Case:

— If the BNF nonterminal is itself defined in the Subclause, then the reference shall be assumed to be the occurrence of that BNF nonterminal on the left side of the defining production.

— Otherwise, the reference shall be assumed to be to a BNF production in which the particular BNF nonterminal is immediately contained.

3.3.4.7 Other terms

Some Syntax Rules define terms, such as *T1*, to denote named or unnamed tables. Such terms are used as table names or correlation names. Where such a term is used as a correlation name, it does not imply that any new correlation name is actually defined for the denoted table, nor does it affect the scopes of any actual correlation names.

An SQL-statement *S1* is said to be executed as a *direct result of executing an SQL-statement* if *S1* is the SQL-statement contained in a <routine> that has been executed, or if *S1* is the value of an <SQL statement variable> referenced by an <execute immediate statement> contained in a <routine> that has been executed, or if *S1* was the value of the <SQL statement variable> that was associated with an <SQL statement name> by a <prepare statement> and that same <SQL statement name> is referenced by an <execute statement> contained in a <routine> that has been executed. An <SQL procedure statement> *S1* is said to be executed as an *indirect result of executing an <SQL procedure statement>* if *S1* is a <triggered SQL statement> that is contained in some <trigger definition> and a triggering <SQL procedure statement> is executed.

3.3.5 Descriptors

A descriptor is a conceptual structured collection of data that defines the attributes of an instance of an object of a specified type. The concept of descriptor is used in specifying the semantics of SQL. It is not necessary that any descriptor exist in any particular form in any database or environment.

Some SQL objects cannot exist except in the context of other SQL objects. For example, columns cannot exist except in tables. Those objects are independently described by descriptors, and the descriptors of enabling objects (e.g., tables) are said to *include* the descriptors of enabled objects (e.g., columns or table constraints). Conversely, the descriptor of an enabled object is said to *be included in* the descriptor of an enabling object.

In other cases, certain SQL objects cannot exist unless some other SQL object exists, even though there is not an inclusion relationship. For example, SQL does not permit an assertion to exist if the tables referenced by the assertion do not exist. Therefore, an assertion descriptor *is dependent on* or *depends on* zero or more table descriptors (equivalently, an assertion *is dependent on* or *depends on* zero or more tables). In general, a descriptor *D1* can be said to depend on, or be dependent on, some descriptor *D2*.

There are two ways of indicating dependency of one construct on another. In many cases, the descriptor of the dependent construct is said to ''include the name of'' the construct on which it is dependent. In this case ''the name of'' is to be understood as meaning 'sufficient information to identify the descriptor of'; thus an implementor might choose to use a pointer or a concatenation of <catalog name>, <schema name>, *et cetera*. Alternatively, the descriptor may be said to include text (*e.g.*, <query expression>, <search condition>). In such cases, whether the implementation includes actual text (with defaults and implications made explicit) or its own style of parse tree is irrelevant; the validity of the descriptor is clearly ''dependent on'' the existence of descriptors for objects that are referred to in it.

The statement that a column ''is based on'' a domain, is equivalent to a statement that a column ''is dependent on'' that domain.

An attempt to destroy a descriptor may fail if other descriptors are dependent on it, depending on how the destruction is specified. Such an attempt may also fail if the descriptor to be destroyed is included in some other descriptor. Destruction of a descriptor results in the destruction of all descriptors included in it, but has no effect on descriptors on which it is dependent.

3.3.6 Index typography

In the Index to this
ANSI American
ISO International
Standard, the following conventions are used:

— Index entries appearing in **boldface** indicate the page where the word, phrase, or BNF nonterminal was defined;

— Index entries appearing in *italics* indicate a page where the BNF nonterminal was used in a Format; and

— Index entries appearing in roman type indicate a page where the word, phrase, or BNF nonterminal was used in a heading, Function, Syntax Rule, Access Rule, General Rule, Leveling Rule, Table, or other descriptive text.

3.4 Object identifier for Database Language SQL

Function

The object identifier for Database Language SQL identifies the characteristics of an SQL-implementation to other entities in an open systems environment.

Format

```
<SQL object identifier> ::=
       <SQL provenance> <SQL variant>

<SQL provenance> ::= <arc1> <arc2> <arc3>

<arc1> ::= iso | 1 | iso <left paren> 1 <right paren>

<arc2> ::= standard | 0 | standard <left paren> 0 <right paren>

<arc3> ::= 9075

<SQL variant> ::= <SQL edition> <SQL conformance>

<SQL edition> ::= <1987> | <1989> | <1992>

<1987> ::= 0 | edition1987 <left paren> 0 <right paren>

<1989> ::= <1989 base> <1989 package>

<1989 base> ::= 1 | edition1989 <left paren> 1 <right paren>

<1989 package> ::= <integrity no> | <integrity yes>

<integrity no> ::= 0 | IntegrityNo <left paren> 0 <right paren>

<integrity yes> ::= 1 | IntegrityYes <left paren> 1 <right paren>

<1992> ::= 2 | edition1992 <left paren> 2 <right paren>

<SQL conformance> ::= <low> | <intermediate> | <high>

<low> ::= 0 | Low <left paren> 0 <right paren>

<intermediate> ::= 1 | Intermediate <left paren> 1 <right paren>

<high> ::= 2 | High <left paren> 2 <right paren>
```

Syntax Rules

1) An <SQL conformance> of <high> shall not be specified unless the <SQL edition> is specified as <1992>.

2) The value of <SQL conformance> identifies the level at which conformance is claimed as follows:

   a) If <SQL edition> specifies <1992>, then

Case:

i) <low>, then Entry SQL level.

ii) <intermediate>, then Intermediate SQL level.

iii) <high>, then Full SQL level.

b) Otherwise:

i) <low>, then level 1.

ii) <intermediate>, then level 2.

3) A specification of <1989 package> as <integrity no> implies that the integrity enhancement feature is not implemented. A specification of <1989 package> as <integrity yes> implies that the integrity enhancement feature is implemented.

4 Concepts

4.1 Data types

A data type is a set of representable values. The logical representation of a value is a <literal>. The physical representation of a value is implementation-dependent.

SQL supports two sorts of data types: *primitive data types* and *abstract data types*. Primitive data types are sometimes called *built-in data types* or *predefined types*. Abstract data types can be defined by a standard, by an implementation, or by an application. Abtract data types have no corresponding <literal>s and thus no logical representation. Abstract data types are summarized in Subclause 11.44, "<abstract data type definition>".

ANSI Only–SQL3

SQL also supports the collection data types of set, multiset and list. A collection is a multiset of values or objects called elements. The elements may be primitive data type elements, abstract data type elements or collections. A set is an unordered collection of distinct elements, with no duplicates. A multiset is similar to a set except that duplicates are permitted. A list is similar to a multiset except that the elements are ordered.

Values corresponding to primitive data types are also primitive in that they have no logical subdivision within this
ANSI American
ISO International
standard.

Values corresponding to abstract data types may be primitive or complex, depending only on the definition of the abstract data type.

Values are either a null value or a non-null value.

A null value is an implementation-dependent special value that is distinct from all non-null values of the associated data type. There is effectively only one null value and that value is a member of every SQL data type. There is no <literal> for a null value although the keyword NULL is used in some places to indicate that a null value is desired.

SQL defines distinct data types named by the following <key word>s: CHARACTER, CHARACTER VARYING, BIT, BIT VARYING, NUMERIC, DECIMAL, INTEGER, SMALLINT, ENUMERATED, FLOAT, REAL, DOUBLE PRECISION, BOOLEAN, DATE, TIME, TIMESTAMP, INTERVAL, and REF.

- 1 paragraph deleted.

For reference purposes, the data types CHARACTER and CHARACTER VARYING, are collectively referred to as character string types. The data types BIT and BIT VARYING are collectively referred to as bit string types. Character string types and bit string types are collectively referred to as string types and values of string types are referred to as strings. The data types NUMERIC,

DECIMAL, INTEGER and SMALLINT are collectively referred to as exact numeric types. The data types FLOAT, REAL, and DOUBLE PRECISION are collectively referred to as approximate numeric types. Exact numeric types and approximate numeric types are collectively referred to as numeric types. Values of numeric type are referred to as numbers. The data types DATE, TIME, and TIMESTAMP are collectively referred to as datetime types. Values of datetime types are referred to as datetimes. The data type INTERVAL is referred to as an interval type. Values of interval types are called intervals.

Each data type has an associated data type descriptor; the contents of a data type descriptor are determined by the specific data type that it describes. A data type descriptor includes an identification of the data type and all information needed to characterize an instance of that data type.

Subclause 6.1, "<data type>", describes the semantic properties of each data type.

Each host language has its own data types, which are separate and distinct from SQL data types, even though similar names may be used to describe the data types. Mappings of SQL data types to data types in host languages are described in Subclause 13.2, "<routine>", and Subclause 21.1, "<embedded SQL host program>". Not every SQL data type has a corresponding data type in every host language.

4.2 Character strings

A character string data type is described by a character string data type descriptor. A character string data type descriptor contains:

— the name of the specific character string data type (CHARACTER or CHARACTER VARYING; NATIONAL CHARACTER and NATIONAL CHARACTER VARYING are represented as CHARACTER and CHARACTER VARYING, respectively);

— the length or maximum length in characters of the character string data type;

— the catalog name, schema name, and character set name of the character set of the character string data type; and

— the catalog name, schema name, and collation name of the collation of the character string data type.

Character sets fall into three categories: those defined by national or international standards, those provided by implementations, and those defined by applications. All character sets, however defined, always contain the <space> character. Character sets defined by applications can be defined to ‘‘reside’’ in any schema chosen by the application. Character sets defined by standards or by implementations reside in the Information Schema (named INFORMATION_SCHEMA) in each catalog, as do collations defined by standards and collations and form-of-use conversions defined by implementations.

The <implementation-defined character repertoire name> SQL_TEXT specifies the name of a character repertoire and implied form-of-use that can represent every character that is in <SQL language character> and all other characters that are in character sets supported by the implementation.

4.2.1 Character strings and collating sequences

A character string is a sequence of characters chosen from the same character repertoire. The character repertoire from which the characters of a particular string are chosen may be specified explicitly or implicitly. A character string has a length, which is the number of characters in the sequence. The length is 0 or a positive integer.

All character strings of a given character repertoire are comparable.

A collating sequence, also known as a collation, is a set of rules determining comparison of character strings in a particular character repertoire. There is a default collating sequence for each character repertoire, but additional collating sequences can be defined for any character repertoire.

Note: A column may be defined as having a default collating sequence. This default collating sequence for the column may be different from the default collating sequence for its character repertoire, e.g., if the <collate clause> is specified in the <column reference>. It will be clear from context when the term ‘‘default collating sequence’’ is used whether it is meant for a column or for a character repertoire.

Given a collating sequence, two character strings are identical if and only if they are equal in accordance with the comparison rules specified in Subclause 8.2, "<comparison predicate>". The collating sequence used for a particular comparison is determined as in Subclause 4.2.3, "Rules determining collating sequence usage".

The <key word>s NATIONAL CHARACTER are used to specify a character string data type with a particular implementation-defined character repertoire. Special syntax (N‘‘string’’) is provided for representing literals in that character repertoire.

A character set is described by a character set descriptor. A character set descriptor includes:

— the name of the character set or character repertoire,

— if the character set is a character repertoire, then the name of the form-of-use,

— an indication of what characters are in the character set, and

— whether or not the character set uses the DEFAULT collation for its character repertoire, and

— if the character set does not utilize the DEFAULT collation for its character repertoire, then the <translation name> conained in the character set’s <translation collation>, if any, the <collation name> contained in the character set’s <collate clause> or <limited collation definition>, if any, and whether or not DESC was specified in the reference to the collation.

For every character set, there is at least one collation. A collation is described by a collation descriptor. A collation descriptor includes:

— the name of the collation,

— the name of the character repertoire on which the collation operates,

— whether the collation has the NO PAD or the PAD SPACE attribute, and

— whether or not this collation utilizes the DEFAULT collation for its character repertoire,

— if the collation does not utilize the DEFAULT collation for its character repertoire, then the <translation name> contained in the collation’s <translation definition>, if any, the <collation name> contained in the collation’s <collation source>, if any, and whether or not DESC was specified in the definition of the collation.

4.2.2 Operations involving character strings

4.2.2.1 Operators that operate on character strings and return character strings

<concatenation operator> is an operator, ||, that returns the character string made by joining its character string operands in the order given.

<character substring function> is a triadic function, SUBSTRING, that returns a string extracted from a given string according to a given numeric starting position and a given numeric length. Truncation occurs when the implied starting and ending positions are not both within the given string.

<fold> is a pair of functions for converting all the lower case characters in a given string to upper case (UPPER) or all the upper case ones to lower case (LOWER), useful only in connection with strings that may contain <simple Latin letter>s.

<form-of-use conversion> is a function that invokes an installation-supplied form-of-use conversion to return a character string *S2* derived from a given character string *S1*. It is intended, though not enforced by this
ANSI American
ISO International
Standard, that *S2* be exactly the same sequence of characters as *S1*, but encoded according some different form-of-use. A typical use might be to convert a character string from two-octet UCS to one-octet Latin1 or *vice versa*.

<trim function> is a function that returns its first string argument with leading and/or trailing pad characters removed. The second argument indicates whether leading, or trailing, or both leading and trailing pad characters should be removed. The third argument specifies the pad character that is to be removed.

<character translation> is a function for changing each character of a given string according to some many-to-one or one-to-one mapping between two not necessarily distinct character sets. The mapping, rather than being specified as part of the function, is some external function identified by a <translation name>.

For any pair of character sets, there are zero or more translations that may be invoked by a <character translation>. A translation is described by a translation descriptor. A translation descriptor includes:

— the name of the translation,

— the name of the character set from which it translates,

— the name of the character set to which it translates, and

— an indication of how the translation is performed.

4.2.2.2 Other operators involving character strings

<length expression> returns the length of a given character string, as an integer, in characters, octets, or bits according to the choice of function.

<position expression> determines the first position, if any, at which one string, *S1*, occurs within another, *S2*. If *S1* is of length zero, then it occurs at position 1 for any value of *S2*. If *S1* does not occur in *S2*, then zero is returned.

<like predicate> uses the triadic operator LIKE (or the inverse, NOT LIKE), operating on three character strings and returning a Boolean. LIKE determines whether or not a character string ''matches'' a given ''pattern'' (also a character string). The characters ''%'' (percent) and ''_'' (underscore) have special meaning when they occur in the pattern. The optional third argument is a character string containing exactly one character, known as the ''escape character'', for use when a percent or underscore is required in the pattern without its special meaning.

4.2.3 Rules determining collating sequence usage

The rules determining collating sequence usage for character strings are based on the following:

— Expressions where no columns are involved (*e.g.*, literals, host variables) are by default compared using the default collating sequence for their character repertoire.

Note: The default collating sequence for a character repertoire is defined in Subclause 10.6, "<character set specification>", and Subclause 11.33, "<character set definition>".

— When columns are involved (*e.g.*, comparing two columns, or comparing a column to a literal), by default the default collating sequence of the columns involved is used so long as the columns have the same default collating sequence.

— When columns are involved having different default collating sequences, explicit specification of the collating sequence in the expression is required via the <collate clause> when the expression participates in a comparison.

— Any explicit specification of collating sequence in an expression overrides any default collating sequence.

To formalize this, <character value expression>s effectively have a coercibility attribute. This attribute has the values *Coercible*, *Implicit*, *No collating sequence*, and *Explicit*. <character value expression>s with the *Coercible*, *Implicit*, or *Explicit* attributes have a collating sequence.

A <character value expression> consisting of a column reference has the *Implicit* attribute, with collating sequence as defined when the column was created. A <character value expression> consisting of a value other than a column (*e.g.*, a host variable or a literal) has the *Coercible* attribute, with the default collation for its character repertoire. A <character value expression> simply containing a <collate clause> has the *Explicit* attribute, with the collating sequence specified in the <collate clause>.

Note: When the coercibility attribute is *Coercible*, the collating sequence is uniquely determined as specified in Subclause 8.2, "<comparison predicate>".

The tables below define how the collating sequence and the coercibility attribute is determined for the result of any monadic or dyadic operation. Table 1, "Collating coercibility rules for monadic operators", shows the collating sequence and coercibility rules for monadic operators, and Table 2, "Collating coercibility rules for dyadic operators", shows the collating sequence and coercibility rules for dyadic operators. Table 3, "Collating sequence usage for comparisons", shows how the collating sequence is determined for a particular comparison.

Table 1—Collating coercibility rules for monadic operators

Operand Coercibility and Collating Sequence		Result Coercibility and Collating Sequence	
Coercibility	**Collating Sequence**	**Coercibility**	**Collating Sequence**
Coercible	default	Coercible	default
Implicit	*X*	Implicit	*X*
Explicit	*X*	Explicit	*X*
No collating sequence		No collating sequence	

Table 2—Collating coercibility rules for dyadic operators

Operand 1 Coercibility and Collating Sequence		Operand 2 Coercibility and Collating Sequence		Result Coercibility and Collating Sequence	
Coercibility	**Collating Sequence**	**Coercibility**	**Collating Sequence**	**Coercibility**	**Collating Sequence**
Coercible	default	Coercible	default	Coercible	default
Coercible	default	Implicit	*Y*	Implicit	*Y*
Coercible	default	No collating sequence		No collating sequence	
Coercible	default	Explicit	*Y*	Explicit	*Y*
Implicit	*X*	Coercible	default	Implicit	*X*
Implicit	*X*	Implicit	*X*	Implicit	*X*
Implicit	*X*	Implicit	$Y \neq X$	No collating sequence	
Implicit	*X*	No collating sequence		No collating sequence	
Implicit	*X*	Explicit	*Y*	Explicit	*Y*
No collating sequence		Any, except Explicit	Any	No collating sequence	
No collating sequence		Explicit	*X*	Explicit	*X*
Explicit	*X*	Coercible	default	Explicit	*X*
Explicit	*X*	Implicit	*Y*	Explicit	*X*
Explicit	*X*	No collating sequence		Explicit	*X*
Explicit	*X*	Explicit	*X*	Explicit	*X*
Explicit	*X*	Explicit	$Y \neq X$	Not permitted: *invalid syntax*	

Table 3—Collating sequence usage for comparisons

Comparand 1 Coercibility and Collating Sequence		Comparand 2 Coercibility and Collating Sequence		
Coercibility	**Collating Sequence**	**Coercibility**	**Collating Sequence**	**Collating Sequence Used For The Comparison**
Coercible	default	Coercible	default	default

Table 3—Collating sequence usage for comparisons (Cont.)

Comparand 1 Coercibility and Collating Sequence		Comparand 2 Coercibility and Collating Sequence		Collating Sequence Used For The Comparison
Coercibility	**Collating Sequence**	**Coercibility**	**Collating Sequence**	
Coercible	default	Implicit	*Y*	*Y*
Coercible	default	No collating sequence		Not permitted: *invalid syntax*
Coercible	default	Explicit	*Y*	*Y*
Implicit	*X*	Coercible	default	*X*
Implicit	*X*	Implicit	*X*	*X*
Implicit	*X*	Implicit	$Y \neq X$	Not permitted: *invalid syntax*
Implicit	*X*	No collating sequence		Not permitted: *invalid syntax*
Implicit	*X*	Explicit	*Y*	*Y*
No collating sequence		Any except Explicit	Any	Not permitted: *invalid syntax*
No collating sequence		Explicit	*X*	*X*
Explicit	*X*	Coercible	default	*X*
Explicit	*X*	Implicit	*Y*	*X*
Explicit	*X*	No collating sequence		*X*
Explicit	*X*	Explicit	*X*	*X*
Explicit	*X*	Explicit	$Y \neq X$	Not permitted: *invalid syntax*

For *n*-adic operations (*e.g.*, <case expression>) with operands X_1, X_2, . . . , X_n, the collating sequence is effectively determined by considering X_1 and X_2, then combining this result with X_3, and so on.

4.3 Bit strings

A bit string is a sequence of bits, each having the value of 0 or 1. A bit string has a length, which is the number of bits in the string. The length is 0 or a positive integer.

A bit string data type is described by a bit string data type descriptor. A bit string data type descriptor contains:

— the name of the specific bit string data type (BIT or BIT VARYING); and

— the length of the bit string data type (in bits).

4.3.1 Bit string comparison and assignment

All bit strings are mutually comparable. A bit string is identical to another bit string if and only if it is equal to that bit string in accordance with the comparison rules specified in Subclause 8.2, "<comparison predicate>".

Assignment of a bit string to a bit string variable is performed from the most significant bit to the least significant bit in the source string to the most significant bit in the target string, one bit at a time.

4.3.2 Operations involving bit strings

4.3.2.1 Operators that operate on bit strings and return bit strings

<bit concatenation> is an operator, ||, that returns the bit string made by concatenating the two bit string operands in the order given.

<bit substring function> is a triadic function identical in syntax and semantics to <character substring function> except that the first argument and the returned value are both bit strings.

4.3.2.2 Other operators involving bit strings

<length expression> returns the length (as an integer number of octets or bits according to the choice of function) of a given bit string.

<position expression> determines the first position, if any, at which one string, *S1*, occurs within another, *S2*. If *S1* is of length zero, then it occurs at position 1 for any value of *S2*. If *S1* does not occur in *S2*, then zero is returned.

4.4 Numbers

A number is either an exact numeric value or an approximate numeric value. Any two numbers are mutually comparable to each other.

A numeric data type is described by a numeric data type descriptor. A numeric data type descriptor contains:

— the name of the specific numeric data type (NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, or DOUBLE PRECISION);

— the precision of the numeric data type;

— the scale of the numeric data type, if it is an exact numeric data type; and

— an indication of whether the precision (and scale) are expressed in decimal or binary terms.

4.4.1 Characteristics of numbers

An exact numeric value has a precision and a scale. The precision is a positive integer that determines the number of significant digits in a particular radix (binary or decimal). The scale is a non-negative integer. A scale of 0 indicates that the number is an integer. For a scale of *S*, the exact numeric value is the integer value of the significant digits multiplied by 10^{-S}.

An approximate numeric value consists of a mantissa and an exponent. The mantissa is a signed numeric value, and the exponent is a signed integer that specifies the magnitude of the mantissa. An approximate numeric value has a precision. The precision is a positive integer that specifies the number of significant binary digits in the mantissa. The value of an approximate numeric value is the mantissa multiplied by 10^{x}, where *x* is the exponent.

Whenever an exact or approximate numeric value is assigned to a data item or parameter representing an exact numeric value, an approximation of its value that preserves leading significant digits after rounding or truncating is represented in the data type of the target. The value is converted to have the precision and scale of the target. The choice of whether to truncate or round is implementation-defined.

An approximation obtained by truncation of a numerical value *N* for an <exact numeric type> *T* is a value *V* representable in *T* such that *N* is not closer to zero than the numerical value of *V* and such that the absolute value of the difference between *N* and the numerical value of *V* is less than the absolute value of the difference between two successive numerical values representable in *T*.

An approximation obtained by rounding of a numerical value *N* for an <exact numeric type> *T* is a value *V* representable in *T* such that the absolute value of the difference between *N* and the numerical value of *V* is not greater than half the absolute value of the difference between two successive numerical values representable in *T*. If there are more than one such values *V*, then it is implementation-defined which one is taken.

All numerical values between the smallest and the largest value, inclusive, representable in a given exact numeric type have an approximation obtained by rounding or truncation for that type; it is implementation-defined which other numerical values have such approximations.

An approximation obtained by truncation or rounding of a numerical value *N* for an <approximate numeric type> *T* is a value *V* representable in *T* such that there is no numerical value representable in *T* and distinct from that of *V* that lies between the numerical value of *V* and *N*, inclusive.

If there are more than one such values *V* then it is implementation-defined which one is taken. It is implementation-defined which numerical values have approximations obtained by rounding or truncation for a given approximate numeric type.

Whenever an exact or approximate numeric value is assigned to a data item or parameter representing an approximate numeric value, an approximation of its value is represented in the data type of the target. The value is converted to have the precision of the target.

Operations on numbers are performed according to the normal rules of arithmetic, within implementation-defined limits, except as provided for in Subclause 6.15, "<numeric value expression>".

4.4.2 Operations involving numbers

As well as the usual arithmetic operators, plus, minus, times, divide, unary plus, and unary minus, there are the following functions that return numbers:

— <position expression> (see Subclause 4.2.2, "Operations involving character strings", and Subclause 4.3.2, "Operations involving bit strings") takes two strings as arguments and returns an integer;

— <length expression> (see Subclause 4.2.2, "Operations involving character strings", and Subclause 4.3.2, "Operations involving bit strings") operates on a string argument and returns an integer;

— <extract expression> (see Subclause 4.7.3, "Operations involving datetimes and intervals") operates on a datetime or interval argument and returns an integer.

4.5 Enumerated types

An enumerated type is a list of distinct identifiers that represents an ordered set of values. All values for a given enumerated type are comparable. Values of two different enumerated types are not comparable.

An enumerated data type is described by an enumerated data type descriptor. An enumerated data type descriptor contains:

— the name of the enumerated data type (ENUMERATED);

— the number of enumeration names that participate in the enumerated type; and

— a list of the enumeration names that participate in the enumerated type.

4.6 Boolean types

A boolean is a data type that may take on the values *true* and *false* .

A boolean data type is described by a boolean data type descriptor. A boolean data type descriptor contains:

— the name of the boolean data type (BOOLEAN).

4.7 Datetimes and intervals

A datetime data type is described by a datetime data type descriptor. An interval data type is described by an interval data type descriptor.

A datetime data type descriptor contains:

— the name of the specific datetime data type (DATE, TIME, TIMESTAMP, TIME WITH TIME ZONE, or TIMESTAMP WITH TIME ZONE); and

— the value of the <time fractional seconds precision>, if it is a TIME, TIMESTAMP, TIME WITH TIME ZONE, or TIMESTAMP WITH TIME ZONE type.

An interval data type descriptor contains:

— the name of the interval data type (INTERVAL);

— an indication of whether the interval data type is a year-month interval or a day-time interval; and

— the <interval qualifier> that describes the precision of the interval data type.

Every datetime or interval data type has an implied *length in positions*. Let *D* denote a value in some datetime or interval data type *DT*. The length in positions of *DT* is constant for all *D*. The length in positions is the number of characters from the character set SQL_TEXT that it would take to represent any value in a given datetime or interval data type.

4.7.1 Datetimes

Table 4, "Fields in datetime items", specifies the fields that can make up a date time value; a datetime value is made up of a subset of those fields. Not all of the fields shown are required to be in the subset, but every field that appears in the table between the first included primary field and the last included primary field shall also be included. If either timezone field is in the subset, then both of them shall be included.

Table 4—Fields in datetime items

Keyword	Meaning
Primary datetime fields	
YEAR	Year
MONTH	Month within year
DAY	Day within month
HOUR	Hour within day
MINUTE	Minute within hour
SECOND	Second and possibly fraction of a second within minute
Timezone datetime fields	
TIMEZONE_HOUR	Hour value of time zone displacement
TIMEZONE_MINUTE	Minute value of time zone displacement

There is an ordering of the significance of <datetime field>s. This is, from most significant to least significant: YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.

The <datetime field>s other than SECOND contain non-negative integer values, constrained by the natural rules for dates using the Gregorian calendar. SECOND, however, can be defined to have a <time fractional seconds precision> that indicates the number of decimal digits maintained following the decimal point in the seconds value, a non-negative exact numeric value.

There are three classes of datetime data types defined within this
ANSI American
ISO International
Standard:

— DATE — contains the <datetime field>s YEAR, MONTH, and DAY;

— TIME — contains the <datetime field>s HOUR, MINUTE, and SECOND; and

— TIMESTAMP — contains the <datetime field>s YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.

Items of type datetime are mutually comparable only if they have the same <datetime field>s.

Datetimes only have absolute meaning in the context of additional information. Time zones are political divisions of the earth's surface that allow the convention that time is measured the same at all locations within the time zone, regardless of the precise value of ''sun time'' at specific locations. Political entities often change the ''local time'' within a time zone for certain periods of the year, e.g., in the summer. However, different political entities within the same time zone are not necessarily synchronized in their local time changes. When a datetime is specified (in SQL-data or elsewhere) it has an implied or explicit time zone specifier associated with it. Unless that time zone specifier, and its meaning, is known, the meaning of the datetime value is ambiguous.

Therefore, datetime data types that contain time fields (TIME and TIMESTAMP) are maintained in Universal Coordinated Time (UTC), with an explicit or implied time zone part.

The time zone part is an interval specifying the difference between UTC and the actual date and time in the time zone represented by the time or timestamp data item. The time zone displacement is defined as

```
INTERVAL HOUR TO MINUTE
```

A TIME or TIMESTAMP that does not specify WITH TIME ZONE has an implicit time zone equal to the local time zone for the SQL-session. The value of time represented in the data changes along with the local time zone for the SQL-session. However, the *meaning* of the time does not change because it is effectively maintained in UTC.

Note: On occasion, UTC is adjusted by the omission of a second or the insertion of a ''leap second'' in order to maintain synchronization with sidereal time. This implies that sometimes, but very rarely, a particular minute will contain exactly 59, 61, or 62 seconds.

4.7.2 Intervals

There are two classes of intervals. One class, called *year-month* intervals, has an express or implied datetime precision that includes no fields other than YEAR and MONTH, though not both are required. The other class, called *day-time* intervals, has an express or implied interval precision that can include any fields other than YEAR or MONTH.

Table 5, "Fields in year-month INTERVAL items", specifies the fields that make up a year-month interval. A year-month interval is made up of a contiguous subset of those fields.

Table 5—Fields in year-month INTERVAL items

Keyword	Meaning
YEAR	Years
MONTH	Months

Table 6, "Fields in day-time INTERVAL items", specifies the fields that make up a day-time interval. A day-time interval is made up of a contiguous subset of those fields.

Table 6—Fields in day-time INTERVAL items

Keyword	Meaning
DAY	Days
HOUR	Hours
MINUTE	Minutes
SECOND	Seconds and possibly fractions of a second

The actual subset of fields that comprise an item of either type of interval is defined by an <interval qualifier> and this subset is known as the precision of the item.

Within an item of type interval, the first field is constrained only by the <interval leading field precision> of the associated <interval qualifier>. Table 7, "Valid values for fields in INTERVAL items", specifies the constraints on subsequence field values.

Table 7—Valid values for fields in INTERVAL items

Keyword	Valid values of INTERVAL fields
YEAR	Unconstrained except by <interval leading field precision>
MONTH	Months (within years) (0-11)
DAY	Unconstrained except by <interval leading field precision>
HOUR	Hours (within days) (0-23)
MINUTE	Minutes (within hours) (0-59)
SECOND	Seconds (within minutes) (0-59.999...)

Values in interval fields other than SECOND are integers. SECOND, however, can be defined to have an <interval fractional seconds precision> that indicates the number of decimal digits maintained following the decimal point in the seconds value.

Fields comprising an item of type interval are also constrained by the definition of the Gregorian calendar.

Year-month intervals are mutually comparable only with other year-month intervals. If two year-month intervals have different interval precisions, they are, for the purpose of any operations between them, effectively converted to the same precision by appending new <datetime field>s to either the most significant end or the least significant end of one or both year-month intervals. New least significant <datetime field>s are assigned a value of 0. When it is necessary to add new most

significant date time fields, the associated value is effectively converted to the new precision in a manner obeying the natural rules for dates and times associated with the Gregorian calendar.

Day-time intervals are mutually comparable only with other day-time intervals. If two day-time intervals have different interval precisions, they are, for the purpose of any operations between them, effectively converted to the same precision by appending new <datetime field>s to either the most significant end or the least significant end of one or both day-time intervals. New least significant <datetime field>s are assigned a value of 0. When it is necessary to add new most significant datetime fields, the associated value is effectively converted to the new precision in a manner obeying the natural rules for dates and times associated with the Gregorian calendar.

****Editor's Note****
The absence in the preceding paragraphs of any discussion of signed intervals has been identified as a Possible Problem. See Possible Problem **112** in the Editor's Notes.

4.7.3 Operations involving datetimes and intervals

Table 8, "Valid operators involving datetimes and intervals", specifies the results of arithmetic expressions involving datetime and interval operands.

Table 8—Valid operators involving datetimes and intervals

Operand 1	Operator	Operand 2	Result Type
Datetime	−	Datetime	Interval
Datetime	+ or −	Interval	Datetime
Interval	+	Datetime	Datetime
Interval	+ or −	Interval	Interval
Interval	* or /	Numeric	Interval
Numeric	*	Interval	Interval

Arithmetic operations involving items of type datetime or interval obey the natural rules associated with dates and times and yield valid datetime or interval results according to the Gregorian calendar.

Operations involving items of type datetime require that the datetime items be mutually comparable. Operations involving items of type interval require that the interval items be mutually comparable.

Operations involving a datetime and an interval preserve the time zone of the datetime operand. If the datetime operand does not include a time zone part, then the local time zone is effectively used.

<overlaps predicate> uses the operator OVERLAPS to determine whether or not two chronological periods overlap in time. A chronological period is specified either as a pair of datetimes (starting and ending) or as a starting datetime and an interval.

<extract expression> operates on a datetime or interval and returns an exact numeric value representing the value of one component of the datetime or interval.

4.8 Object identifier type

An object identifier *OID* is a value generated when an object is created, to give that object an immutable identity. It is unique in the known universe of objects that are instances of abstract data types, and is conceptually separate from the value, or state, of the instance.

The object identifier type is described by an object identifier type descriptor. An object identifier type descriptor contains:

— the name of the object identifier type (REF).

— the name of the abstract data type in case it is further qualified by one.

Conceptually, every object has an OID, although it is possible to specify as an aid to optimization that an ADT has all its instances without OID, if the OID need never be used.

ANSI Only–SQL3

An OID literal exists for an object identifier type only if the associated abstract data type was defined WITH OID VISIBLE. The OID value is materialized as a character string with an implementation-defined length and character set SQL_TEXT.

ISO Only—caused by ANSI changes not yet considered by ISO

As with other data types, it is necessary to materialize an OID only when it is passed to a standard programming language. If this is not needed, another possible aid to optimization is to specify the ADT with OID ‘‘not visible’’. In the case with OID ‘‘visible’’, the OID is materialized as a character string with an implementation-defined length and character set SQL_TEXT.

4.9 Abstract data types

An abstract data type (ADT) is a data type whose characteristics are specified by an abstract data type descreiptor. An abstract data type descriptor specifies a set of declarations of the stored attributes that represent the value of the abstract data type, the operations that define the equality and ordering relationships of the abstract data type, and the operations and derived attributes that implement the behavior of the abstract data type. The operations and derived attributes are implemented by <routine>s (see Subclause 13.2, "<routine>" and Subclause 10.3, "<routine invocation>"). A value of an abstract data type is comparable only to another value of the same abstract data type.

An abstract data type is described by an abstract data type descriptor. An abstract data type descriptor contains:

— the name of the abstract data type;

— the name of the equals operation for the abstract data type (or DEFAULT, as appropriate);

— the name of the less-than operation for the abstract data type (or NONE or REF, as appropriate);

— if the abstract data type has one or more cast operations, then the names of each of those cast operations;

— if the abstract data type is a subtype of one or more other abstract data types, then the names of those abstract data types;

— the column descriptor of every attribute of the abstract data type;

— the descriptor of each table constraint specified for the type;

— The degree of the abstract data type (the number of column descriptors);

— an indication of whether the abstract data type is WITH OID VISIBLE, WITH OID NOT VISIBLE, or WITHOUT OID;

— the function descriptor of every operation that has the abstract data type as a parameter or result;

— other information To Be Supplied.

4.9.1 Encapsulation

Each component of an abstract data type has its encapsulation level specified as either public, private, or protected. Public components form the interface of the ADT and are visible to all authorized users of the ADT. Private components are totally encapsulated and are visible only within the definition of the ADT that contains them. Protected components are partially encapsulated, being visible both within their own ADT and within the definitions of all subtypes of that ADT.

Encapsulation is a property of the structure of the ADT definitions, independently of questions of which users are privileged to create or use ADTs (see Subclause 4.35, "Privileges and roles"). Once it is determined what components are public and are thus in principle accessible outside an ADT definition, the privilege mechanisms are applied to control whether a given user is authorized to use a given public component.

4.9.2 Subtypes and supertypes

A subtype is a named type described by an abstract data type descriptor that contains a <subtype clause>. If type T_a is declared to be a subtype of another type T_b by an <abstract data type definition>, then T_a is called a *direct subtype* of T_b. The subtype is defined as follows:

Type T_a is a subtype of type T_b if and only if either:

— T_a and T_b are the same named type,

— T_a is a direct subtype of T_b, or

— there is a type T_c such that T_a is a direct subtype of T_c and T_c is a subtype of T_b.

A type *T* is considered to be one of its own subtypes. Subtypes of *T* other than *T* itself are called its *proper subtypes*.

Type T_b is called a *supertype* of T_a if T_a is a subtype of T_b. If T_a is a direct subtype of T_b, then T_b is called a *direct supertype* of T_a. A type that is not a subtype of any other type is called a *maximal supertype*. A supertype shall not have itself as a proper subtype.

Let T_a be a maximal supertype and let *T* be a subtype of T_a. The set of all subtypes of T_a (which includes T_a itself) is called a *subtype family* of *T* or (equivalently) of T_a. There shall be only one maximal supertype in a subtype family.

Users must have the UNDER privilege on a type before they can use the type in a subtype definition. A type can have more than one proper subtype. Similarly, a type can have more than one proper supertype. Thus, a subtype is a specialized type of one or more supertypes and a supertype is a generalized type of one or more subtypes.

Any instance of a subtype is an instance of all of its supertypes. An instance is associated with exactly one *most specific type*, which is defined by the property that every other type associated with the instance is a supertype of the most specific type. The most specific type of an instance need not correspond to a leaf type in the type hierarchy. For example, a type hierarchy might consist of a maximal supertype PERSON which has STUDENT and EMPLOYEE as its two subtypes, while STUDENT has two direct subtypes UG_STUDENT and PG_STUDENT. An instance in this hierarchy might be created with a most specific type of STUDENT, even though it does not correspond to a leaf type in the hierarchy.

If T_a is a subtype of T_b, then an instance of T_a can be used wherever an instance of T_b is expected, *i.e.*, it is always possible to cast the instance of T_a to an instance of T_b. In particular, if *O* is an operation with parameter *P* of type T_b, then an invocation of *O* can use an instance of type T_a as the actual parameter *P*.

A subtype definition has lexical access to the representation of all of its direct supertypes (but only within the ADT definition that defines the subtype of that supertype), but it has no lexical access to the representation of its sibling types. Effectively, components of all direct supertype representations are copied to the subtype's representation with same name and data type. To avoid name clashes, a subtype can rename selected components of the representation inherited from its direct supertypes.

A subtype can define constructor, actor, and destructor operations like any other ADT. A subtype can also define operations which have the same name as operations defined on other types, including its supertypes.

4.9.3 Type templates

A type template is used to define a family of abstract data types. Each type template definition specifies a set of formal parameters and a body. The parameters specify values or data types that must be provided when the type template is used to designate a specific generated type. The type template body is identical to that of an abstract data type definition, except that it may contain the formal parameters of the type template. A type template is described by a type template descriptor which contains:

— the name of the type template,

— the name, <template parameter type>, and position of each of the type template's parameters, and

— the body of the type template.

A generated type is an abstract data type resulting from a <generated type reference>. A <generated type reference> specifies the type template name and a set of actual parameters. Each actual parameter must be a <literal>, a <data type>, or some other formal parameter whose actual parameter (directly or indirectly) is in turn a <literal> or <data type>.

A <generated type reference> may be specified wherever a <data type> is allowed; thus nesting of <generated type reference>s is also allowed.

Type template names may be overloaded; that is, more than one type template with the same name may be defined. To define such type templates, their parameter declaration lists must differ sufficiently such that it's possible to unambiguously determine for each <generated type reference>, which type template it references.

ANSI Only–SQL3

4.10 Collection types

A collection is a multiset of objects or values called elements. All elements in a collection have the same <data type>, and each <collection type> specifies that <data type>.

A collection has the semantics of a table, as described in Subclause 7.13, "<query expression>". Likewise, a table has the semantics of a collection.

Each element of a collection corresponds to a row in the table. For a collection of ADT elements, each attribute of the ADT type corresponds to a column in the table. For a collection of elements of a <predefined type> or <collection type>, the collection table has a single column of that <predefined type> or <collection type>. A collection is a value; that is, a collection has no OID.

Because a collection has the semantics of a table, the operations defined on tables (*e.g.*, INTERSECTION) also apply to collections. Wherever the rules of this standard specify ''table'', those rules also apply to a collection. Wherever the rules of this standard specify ''row'', those rules also apply to the corresponding element in the collection. A collection type is a set type, multiset type or list type.

A collection type is described by a collection type descriptor that includes:

— The name of the data type of the elements;

— An indication of whether the collection is a set, multiset, or list.

4.10.1 Set type

A set type is a collection type that has the semantics of a mathematical set: a set is an unordered collection of distinct elements.

A set value with elements of type *DT* has the data type SET(*DT*). An empty set (a set with no elements) has the data type SET( ).

Two set values are comparable if the data types of their elements are the same or one or both set values have the data type SET( ).

Set values are assignable if the source value has the same data type as the data type of the target item or has SET( ) data type. A set value of type SET(*DT*) retains its data type even when it becomes an empty set.

Two sets are equal if they contain the same elements and those elements are all non-null values. The equality of two sets is unknown if the sets contain null values and the replacement of those null value with appropriate non-null values would make the two sets equal. Otherwise, two sets are unequal.

For any data type *DT*, a set type of *DT* elements is a subtype of a multiset type of *DT* elements.

4.10.2 Multiset type

The multiset type is a collection type that is similar to the set type, except that duplicate elements are permitted in a multiset.

A multiset value with elements of type *DT* has the data type MULTISET(*DT*). An empty multiset (a multiset with no elements) has the data type MULTISET( ).

Two multiset values are comparable if the data types of their elements are the same or one or both multiset values have the data type MULTISET( ).

Multiset values are assignable if the source value has the same data type as the data type of the target item or has MULTISET( ) data type. A multiset value of type MULTISET(*DT*) retains its data type even when it becomes an empty multiset. Two multisets are equal if they contain the same elements and those elements are all non-null values. The equality of two multisets is unknown if the multisets contain null values and the replacement of those null value with appropriate non-null values would make the two multisets equal. Otherwise, two multisets are unequal.

4.10.3 List type

A list type is a collection type that is similar to the multiset type, except that the elements of a list are ordered. Thus, when an element is inserted into a list, the insert operation may specify the location in the list where the new element is to be inserted.

A list value with elements of type *DT* has the data type LIST(*DT*). An empty list (a list with no elements) has the data type LIST( ).

Two list values are comparable if the data types of their elements are the same or one or both list values have the data type LIST( ).

List values are assignable if the source value has the same data type as the data type of the target item or has LIST( ) data type. A list value of type LIST(*DT*) retains its data type even when it becomes an empty list.

Two lists are equal if they contain the same elements in the same order and those elements are all non-null values. The equality of two lists is unknown if the lists contain null values and the replacement of those null value with appropriate non-null values would make the two lists equal. Otherwise, two lists are unequal.

For any data type *DT*, a list type of *DT* elements is a subtype of a multiset type of *DT* elements.

4.11 Type conversions and mixing of data types

Values of the data types NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, and DOUBLE PRECISION are numbers and are all mutually comparable and mutually assignable. If an assignment would result in a loss of the most significant digits, an exception condition is raised. If least significant digits are lost, implementation-defined rounding or truncating occurs with no exception condition being raised. The rules for arithmetic are generally governed by Subclause 6.15, "<numeric value expression>".

****Editor's Note****
The wording of the preceding paragraph has been identified as a Possible Problem. See Possible Problem **104** in the Editor's Notes.

Values corresponding to the data types CHARACTER and CHARACTER VARYING are mutually assignable if and only if they are taken from the same character repertoire. If they are from different character repertoires, then the value of the source of the assignment must be translated to the character repertoire of the target before an assignment is possible. If a store assignment would result in the loss of non-<space> characters due to truncation, then an exception condition is raised. The values are mutually comparable only if they are mutually assignable and can be coerced to have the same collation. The comparison of two character strings depends on the collating sequence used for the comparison (see Table 3, "Collating sequence usage for comparisons"). When values of unequal length are compared, if the collating sequence for the comparison has the NO PAD attribute and the shorter value is equal to a prefix of the longer value, then the shorter value is considered less than the longer value. If the collating sequence for the comparison has the PAD SPACE attribute, for the purposes of the comparison, the shorter value is effectively extended to the length of the longer by concatenation of <space>s on the right.

Values corresponding to the data types BIT and BIT VARYING are always mutually comparable and are mutually assignable. If a store assignment would result in the loss of bits due to truncation, then an exception condition is raised. When values of unequal length are to be compared, if the shorter is a prefix of the longer, then the shorter is less than the longer; otherwise, the longer is effectively truncated to the length of the shorter for the purposes of comparison. When values of equal length are to be compared, then a bit-by-bit comparison is made. A 0-bit less than a 1-bit.

The boolean data type is discussed in Subclause 4.6, "Boolean types".

Values of type datetime are mutually assignable only if the source and target of the assignment have the same datetime fields.

Values of type interval are mutually assignable only if the source and target of the assignment are both year-month intervals or if they are both day-time intervals.

Values corresponding to abstract data types are mutually assignable and mutually comparable if both have the same abstract data type.

Values corresponding to object identifier types are comparable, using the operators ''equals'' and ''not equals'' only. They are assignable if a value is to be stored in an item whose type is REF without being further qualified by an abstract data type name. They are also assignable if a value that is of type REF*T1* is to be stored in an item that is of type REF*T2* and *T1* is the same as *T2* or *T1* is a subtype of *T2*.

A value corresponding to an abstract data type is not comparable to a value of object identifier type. Explicit REF or DEREF functions must be used to make both values of abstract data type or both values of object identifier type.

A value corresponding to an abstract data type *T1* is assignable to an item of object identifier type REF(*T2*) if *T1* is specified WITH OID and, if by applying the REF function to the value, the resulting object identifier is assignable by the rules above.

A value corresponding to an abstract data type REF(*T1*) is assignable to an item of abstract data type *T2* if, by applying the DEREF function to the value, the resulting instance of *T1* is assignable, *i.e.*, if *T1* is the same abstract data type as *T2*.

ANSI Only–SQL3

Values corresponding to the collection types are discussed in Subclause 4.10, "Collection types".

Implicit type conversion can occur in expressions, fetch operations, single row select operations, inserts, deletes, and updates. Explicit type conversions can be specified by the use of the CAST operator.

****Editor's Note****

The fact that this Subclause doesn't really talk about conversions for all data types has been identified as a Possible Problem. See Possible Problem **107** in the Editor's Notes.

4.12 Domains

A domain is a set of permissible values. A domain is defined in a schema and is identified by a <domain name>. The purpose of a domain is to constrain the set of valid values that can be stored in SQL-data by various operations.

A domain definition specifies a data type. It may also specify a <domain constraint> that further restricts the valid values of the domain and a <default clause> that specifies the value to be used in the absence of an explicitly specified value or column default.

A domain is described by a domain descriptor. A domain descriptor includes:

— the name of the domain;

— the data type descriptor of the data type of the domain;

— the <collation name> from the <collate clause>, if any, of the domain;

— the null class of the domain;

— the value of <default option>, if any, of the domain; and

— the domain constraint descriptors of the domain constraints, if any, of the domain.

4.13 Nulls

Every domain or column in a base
ANSI table, and every SQL variable and SQL-supplied parameter,
ISO table
has a null class. If no null class is specified, it is the general null class (which contains only the general null value); otherwise it is the defined null class that is specified.

A defined null class, is created by a <null class definition> and is a named set of possible null values known as null states, together with the general null value. A null state is a named, implementation-dependent null value that is distinct from both the general null value and all other null states of the same null class.

The null values of a null class are ordered on their position number, the general null value having position number one in every null class. This ordering is used to determine the result of an operation when more than one of its operands are null values.

Except in the few cases where the null substitution principle yields a different result, if one or more operands of an expression is a null value then the result is a null value.

The result of an attempted transfer of any null value between objects having different null classes is the general null value.

The null class of the result of an operation is determined as follows.

Case:

— If no operand is null, then the result is as determined by the application of other General Rules.

— If the operator is AND and either operand is false, then the result is false.

— If the operator is OR and either operand is true, then the result is true.

— If the result has a defined null class, and one or more operands are null, then:

  Case:

  * If the operator is OR then the result is the null value having the maximum position number;

  * Otherwise, the result is the null value having the minimum position number.
    Note: The general null value effectively has the position number one.

— Otherwise, the result is the general null value.

4.14 Columns and attributes

A column is a multiset of values that may vary over time. All values of the same column are of the same data type or domain and are values in the same table. A value of a column is the smallest unit of data that can be selected from a table and the smallest unit of data that can be updated. If a table is a collection of instances of an abstract data type, then a column of the table is a multiset of the values of an attribute of that ADT and has the name of that attribute.

Every column has a <column name>.

Every stored column or stored attribute has a nullability characteristic of *known not nullable* or *possibly nullable*, defined as follows:

A column or stored attribute has a nullability characteristic that indicates whether any attempt to store a null value into that column or stored attribute will inevitably raise an exception, or whether any attempt to retrieve a value from that column or stored attribute can ever result in a null value.
A stored column *C* of a base table *T* or a stored attribute *C* of an abstract data type
ANSI *T*, with the column name of *C* being *CN*,
ISO *T*
is *known not nullable* if and only if either:

— there exists at least one constraint that is not deferrable and that simply contains a <search condition> that contains *CN* IS NOT NULL or NOT *CN* IS NULL or *RVE* IS NOT NULL, where *RVE* is a <row value constructor> that contains a <row value constructor element> that is simply *CN* without an intervening <search condition> that specifies OR and without an intervening <boolean factor> that specifies NOT.

— *C* is based on a domain that has a domain constraint that is not deferrable and that simply contains a <search condition> that contains VALUE IS NOT NULL or NOT VALUE IS NULL without an intervening <search condition> that specifies OR and without an intervening <boolean factor> that specifies NOT.

— *C* is a unique column of a nondeferrable unique constraint that is a PRIMARY KEY.

Otherwise, a stored column or stored attribute *C* is *possibly nullable*.

A column or attribute, *C*, is described by a column descriptor or attribute descriptor, respectively. A column or attribute descriptor includes:

— the name of the column;

— whether the name of the column is an implementation-dependent name;

— if the column is based on a domain, then the name of that domain; otherwise, the data type descriptor of the data type of *C* and the null class of *C*;

— the <collation name> from the <collate clause>, if any, of C;

— an indication of whether *C* is stored or virtual;

— for a stored column, the value of <default option>, if any, of *C*;

— for a stored column, the nullability characteristic of *C*;

— for a virtual column, the name of the get function and the name of the set function (if any) of *C*;

— an indication of whether the column is updatable, constant, or read-only;

— the ordinal position of *C* within the table or abstract data type that contains it; and

— the encapsulation level of *C*.

ANSI Only–SQL3

4.15 Tables, sets, and lists

A table is a collection of elements that are known as its rows. A row has a row type that is either an abstract data type or a predefined type, and every row in a table is of the same row type. The row is the smallest unit of data that can be inserted into a table and deleted from a table.

In general, a table is a multiset of rows, *i.e.*, a collection that may contain duplicate rows. A set is treated as a special case of a table that contains no duplicate rows, *i.e.*, it is a set of rows. A list is a special case of a table that also has an ordering of its elements, *i.e.*, it is a list of rows. Thus, the term ''table'' is used for any collection of rows, including multisets that have the additional properties of sets or lists as well as those that do not.

If a row is an instance of an abstract data type, then the value of the row is the value of that instance, *i..e*, a sequence of values of the attributes of the type, and the *i*-th value in every row of the table is a value of the *i*-th column of that table.

If a row is an instance of a predefined type, then the value of the row is the value of that instance, *i.e.*, a single value of that predefined type, not subdivided into attributes in columns. However, a column descriptor is also generated with the special column name ''CONTENTS'' and the predefined type as its data type, so that the table can be treated as a single-column table where the context requires this, as in joining one table to another or in using the definition of the degree of a table given below.

Since a table is a collection of objects of a data type, a table can be used in any context where a collection of elements of that data type is permitted. Likewise, a collection can be used in any context where a table expression is permitted. Just as every collection is either a set, a multiset, or a list, so every table is either a set table, a multiset table, or a list table.

ISO Only—caused by ANSI changes not yet considered by ISO

4.16 Tables

A table is a multiset of rows. A row is a nonempty sequence of values. (If the row is an instance of an abstract data type, then this sequence is the sequence of values of the attributes of the type.) Every row of the same table has the same cardinality and contains a value of every column of that table. The *i*-th value in every row of a table is a value of the *i*-th column of that table. The row is the smallest unit of data that can be inserted into a table and deleted from a table.

ANSI Only–SQL3

Sets and lists are treated as specialized tables. They share all the properties of tables, but have the additional properties that:

— For a set, the table can contain no duplicate rows;

— For a list, there is an ordering of the rows of the table.

The degree of a table is the number of columns of that table. At any time, the degree of a table is the same as the cardinality of each of its rows and the cardinality of a table is the same as the cardinality of each of its columns. A table whose cardinality is 0 is said to be *empty*.

A table is either a base table, a viewed table, or a derived table. A base table is either a persistent base table, a global temporary table, a created local temporary table, or a declared local temporary table.

A viewed table is either a permanent viewed table or a temporary viewed table. A permanent viewed table is called a *view* or a *viewed table*; a temporary viewed table is called a *temporary view*.

A persistent base table is a named table defined by a <table definition> that does not specify TEMPORARY.

A derived table is a table derived directly or indirectly from one or more other tables by the evaluation of a <query expression>. The values of a derived table are derived from the values of the underlying tables when the <query expression> is evaluated.

A viewed table is a named derived table defined by a <view definition>. A viewed table is sometimes called a *view*.

The terms *simply underlying table*, *underlying table*, *leaf underlying table*, *generally underlying table*, and *leaf generally underlying table* define a relationship between a derived table or cursor and other tables.

The *simply underlying table*s of derived tables and cursors are defined in Subclause 7.12, "<query specification>", Subclause 7.13, "<query expression>", and Subclause 15.1, "<declare cursor>". A viewed table has no *simply underlying table*s.

The *underlying table*s of a derived table or cursor are the simply underlying tables of the derived table or cursor and the underlying tables of the simply underlying tables of the derived table or cursor.

The *leaf underlying table*s of a derived table or cursor are the underlying tables of the derived table or cursor that do not themselves have any underlying tables.

The *generally underlying table*s of a derived table or cursor are the underlying tables of the derived table or cursor and, for those underlying tables of the derived table or cursor that are viewed tables, the <query expression> of each viewed table and the generally underlying tables of the <query expression> of each viewed table.

The *leaf generally underlying table*s of a derived table or cursor are the generally underlying tables of the derived table or cursor that do not themselves have any generally underlying tables.

All base tables are updatable. Derived tables are either updatable or read-only. The operations of insert, update, and delete are permitted for updatable tables, subject to constraining Access Rules. The operations of insert, update, and delete are not allowed for read-only tables.

A grouped table is a set of groups derived during the evaluation of a <group by clause> or a <having clause>. A group is a multiset of rows in which all values of the grouping column or columns are equal if a <group by clause> is specified, or the group is the entire table if no <group by clause> is specified. A grouped table may be considered as a collection of tables. Set functions may operate on the individual tables within the grouped table.

A global temporary table is a named table defined by a <table definition> that specifies GLOBAL TEMPORARY. A created local temporary table is a named table defined by a <table definition> that specifies LOCAL TEMPORARY. Global and created local temporary tables are effectively materialized only when referenced in an SQL-session. Every <module> in every SQL-session that references a created local temporary table causes a distinct instance of that created local temporary table to be materialized. That is, the contents of a global temporary table or a created local temporary table cannot be shared between SQL-sessions. In addition, the contents of a created local temporary table cannot be shared between <module>s of a single SQL-session. The definition of a global temporary

table or a created local temporary table appears in a schema. In SQL language, the name and the scope of the name of a global temporary table or a created local temporary table are indistinguishable from those of a persistent base table. However, because global temporary table contents are distinct within SQL-sessions, and created local temporary tables are distinct within <module>s within SQL-sessions, the *effective* <schema name> of the schema in which the global temporary table or the created local temporary table is instantiated is an implementation-dependent <schema name> that may be thought of as having been effectively derived from the <schema name> of the schema in which the global temporary table or created local temporary table is defined and the implementation-dependent SQL-session identifier associated with the SQL-session. In addition, the *effective* <schema name> of the schema in which the created local temporary table is instantiated may be thought of as being further qualified by a unique implementation-dependent name associated with the <module> in which the created local temporary table is referenced.

ANSI Only–SQL3

A declared local temporary table is either a module local temporary table or a compound statement local temporary table. A module local temporary table is a named table defined by a <temporary table declaration> in a <module>. A compound statement local temporary table is a named table defined by a <temporary table declaration> in a <compound statement>. A module local temporary table is effectively materialized the first time it is referenced in an SQL-session, and it persists for that SQL-session. A compound statement local temporary table is effectively materialized the first time it is referenced during an execution of the compound statement, and it persists for that execution.

ISO Only—caused by ANSI changes not yet considered by ISO

A declared local temporary table is a named table defined by a <temporary table declaration> that is effectively materialized the first time any <routine> in the <module> that contains the <temporary table declaration> is executed. A declared local temporary table is accessible only by <routine>s in the module that contains the <temporary table declaration>. The effective <schema name> of the <schema qualified name> of the declared local temporary table may be thought of as the implementation-dependent SQL-session identifier associated with the SQL-session and a unique implementation-dependent name associated with the <module> that contains the <temporary table declaration>. All references to a declared local temporary table are prefixed by ''MODULE.''.

The materialization of a temporary table does not persist beyond the end of the SQL-session in which the table was materialized. Temporary tables are effectively empty at the start of an SQL-session.

A subtable is a named table defined by a <table definition> that contains a <subtable clause>. If table T_a is declared to be a subtable of another table T_b by a <table definition>, T_a is called a *direct subtable* of T_b. The subtable is defined as follows:

Table T_a is a subtable of T_b if one of the following holds,

1) T_a is a direct subtable of T_b.

2) There is a table T_c such that T_a is a direct subtable of T_c and T_c is a subtable of T_b.

Table T_b is called a supertable of T_a if T_a is a subtable of T_b. If T_a is a direct subtable of T_b, then T_b is called a *direct supertable* of T_a. A supertable that is not a subtable of any other table is called a *maximal supertable*. A supertable shall not have itself as a subtable.

A subtable is a specialized table of one or more supertables, and a supertable is a generalized table of one or more subtables. Supertables and subtables are base tables. Supertables and subtables may have the same rows.

When a subtable is defined, the subtable inherits every column from its supertables. The phrase *inherited column* denotes a column that is inherited from some supertable. The phrase *originally defined column* denotes a column that is specified in the <table definition> of the subtable.

Let T_a be a maximal supertable and T be a subtable of T_a. The set T_a and all subtables of T_a is called a *subtable family* of T or of T_a. Given a row of a maximal supertable T_a, the set of rows which each have the same primary key value and belong to a table in the corresponding subtable family is called a *subtable row family*. There shall be only one maximal supertable in a subtable family.

Any row of a subtable must correspond to one and only one row of each direct supertable. Any row of a supertable corresponds to at most one row of a direct subtable. Two column values of two rows belonging to the same subtable row family coincide if the two columns are inherited from the same column.

A temporary viewed table is a named derived table defined by a <temporary view definition>. A temporary view does not persist across SQL-sessions. A temporary view definition appears within a <module>. Whether a temporary viewed table is materialized is implementation-dependent.

A table is described by a table descriptor. A table descriptor is either a base table descriptor, a view descriptor, or a derived table descriptor (for a derived table that is not a view).

Every table descriptor includes:

ANSI Only–SQL3

— an indication of whether the table was created as a multiset table (an ordinary persistent base table), a set table, or a list table;

— the degree of the table (the number of column descriptors); and

— the column descriptor of each column in the table.

A base table descriptor describes a base table. In addition to the components of every table descriptor, a base table descriptor includes:

— the name of the base table;

— an indication of whether the table is a persistent base table, a global temporary table, a created local temporary table, or a declared local temporary table;

— the descriptor of each table constraint specified for the table;

ANSI Only–SQL3

— The name of the data type of the objects in the table; if the table is defined by an abstract data type, then this is the name of that abstract data type; otherwise, this is an implementation-dependent data type name;

ISO Only—caused by ANSI changes not yet considered by ISO

— if the table is defined by an abstract data type, then the name of that abstract data type; otherwise, the degree of the table (the number of column descriptors) and the column descriptor of each column in the table;

— if the table is a subtable, then the names of its direct supertables;

ISO Only—caused by ANSI changes not yet considered by ISO

— if the table was defined to be OF <abstract data type name>, then the name of that abstract data type;

— a non-empty set of functional dependencies, according to the rules given in Subclause 4.18, "Functional dependencies and candidate keys";

— a non-empty set of candidate keys; and

— a preferred candidate key, which may or may not be additionally designated the primary key, according to the Rules in Subclause 4.18, "Functional dependencies and candidate keys".

A derived table descriptor describes a derived table. In addition to the components of every table descriptor, a derived table descriptor includes:

— if the table is named, then the name of the table;

— the <query expression> that defines how the table is to be derived; and

— an indication of whether the derived table is updatable or
ANSI read-only;
ISO read-only (this is derived from the <query expression>);

A view descriptor describes a view. In addition to the components of a derived table descriptor, a view descriptor includes:

— an indication of whether the view has the CHECK OPTION; if so, whether it is to be applied as CASCADED or LOCAL; and

— an indication of whether it is a persistent view or a temporary view.

4.17 Integrity constraints

Integrity constraints, generally referred to simply as constraints, define the valid states of SQL-data by constraining the values in the base tables. A constraint is either a table constraint, a domain constraint or an assertion. A constraint is described by a constraint descriptor. A constraint descriptor is either a table constraint descriptor, a domain constraint descriptor or an assertion descriptor. Every constraint descriptor includes:

— the name of the constraint;

— an indication of whether or not the constraint is deferrable;

— an indication of whether the initial constraint mode is *deferred* or *immediate*;

A <query expression> or <query specification> is *possibly non-deterministic* if an implementation might, at two different times where the state of the SQL-data is the same, produce results that differ by more than the order of the rows due to General Rules that specify implementation-dependent behavior.

No integrity constraint shall be defined using a <query specification> or a <query expression> that is possibly non-deterministic.

4.17.1 Checking of constraints

Every constraint is either *deferrable* or *non-deferrable*. Within a transaction, every constraint has a constraint mode; if a constraint is *non-deferrable*, then its constraint mode is always *immediate*, otherwise it is either or *immediate* or *deferred*. Every constraint has an initial constraint mode that specifies the constraint mode for that constraint at the start of each SQL-transaction and immediately after definition of that constraint. If a constraint is *deferrable*, then its constraint mode may be changed (from *immediate* to *deferred*, or from *deferred* to *immediate*) by execution of a <set constraints mode statement>.

The checking of a constraint depends on its constraint mode within the current SQL-transaction. If the constraint mode is *immediate*, then the constraint is effectively checked at the end of each SQL-statement. If the constraint mode is *deferred*, then the constraint is effectively checked when the constraint mode is changed to *immediate* either explicitly by execution of a <set constraints mode statement>, or implicitly at the end of the current SQL-transaction.

When a constraint is checked other than at the end of an SQL-transaction, if it is not satisfied, then an exception condition is raised and the SQL-statement that caused the constraint to be checked has no effect other than entering the exception information into the diagnostics area. When a <commit statement> is executed, all constraints are effectively checked and, if any constraint is not satisfied, then an exception condition is raised and the transaction is terminated by an implicit <rollback statement>.

4.17.2 Table constraints

A table constraint is either a unique constraint, a referential constraint or a table check constraint. A table constraint is described by a table constraint descriptor which is either a unique constraint descriptor, a referential constraint descriptor or a table check constraint descriptor.

A unique constraint is described by a unique constraint descriptor. In addition to the components of every table constraint descriptor, a unique constraint descriptor includes:

— an indication of whether it was defined with PRIMARY KEY or UNIQUE, and

— the names and positions of the *unique columns* specified in the <unique column list>;

A referential constraint is described by a referential constraint descriptor. In addition to the components of every table constraint descriptor, a referential constraint descriptor includes:

— the names of the *referencing columns* specified in the <referencing columns>,

— the names of the *referenced columns* and *referenced table* specified in the <referenced table and columns>, and

— an indication of whether PENDANT was specified,

— the value of the <match type>, if specified, and the <referential triggered actions>, if specified.

Note: If MATCH FULL or MATCH PARTIAL is specified for a referential constraint and if the referencing table has only one column specified in <referential constraint definition> for that referential constraint, or if the referencing table has more than one specified column for that <referential constraint definition>, but none of those columns is nullable, then the effect is the same as if no <match option> were specified.

A table check constraint is described by a table check constraint descriptor. In addition to the components of every table constraint descriptor, a table check constraint descriptor includes:

— the <search condition>.

A unique constraint is satisfied if and only if no two rows in a table have the same non-null values in the *unique columns*. In addition, if the unique constraint was defined with PRIMARY KEY, then it requires that none of the values in the specified column or columns be a null value.

In the case that a table constraint is a referential constraint, the table is referred to as the *referencing table*. The *referenced columns* of a referential constraint shall be the *unique columns* of some unique constraint of the *referenced table*.

A referential constraint is satisfied if one of the following conditions is true, depending on the <match option> specified in the <referential constraint definition>:

— If no <match type> was specified then, for each row *R1* of the *referencing table*, either at least one of the values of the *referencing columns* in *R1* shall be a null value, or the value of each referencing column in *R1* shall be equal to the value of the corresponding *referenced column* in some row of the *referenced table*.

— If MATCH FULL was specified then, for each row *R1* of the *referencing table*, either the value of every *referencing column* in *R1* shall be a null value, or the value of every *referencing column* in *R1* shall not be null and there shall be some row *R2* of the *referenced table* such that the value of each *referencing column* in *R1* is equal to the value of the corresponding *referenced column* in *R2*.

— If MATCH PARTIAL was specified then, for each row *R1* of the *referencing table*, there shall be some row *R2* of the *referenced table* such that the value of each *referencing column* in *R1* is either null or is equal to the value of the corresponding *referenced column* in *R2*.

The referencing table may be the same table as the referenced table.

A table check constraint is satisfied if and only if the specified <search condition> is not false for any row of a table.

A PENDANT specification further requires that for each row of the *referenced table*, the values of the specified column or columns are the same as the values of the specified column or columns in some row of some *referencing table*.

4.17.3 Domain constraints

A domain constraint is a constraint that is specified for a domain. It is applied to all columns that are based on that domain, and to all values cast to that domain.

A domain constraint is described by a domain constraint descriptor. In addition to the components of every constraint descriptor a domain constraint descriptor includes:

— the <search condition>.

A domain constraint is satisfied by SQL-data if and only if, for any table *T* that has a column named *C* based on that domain, the specified <search condition>, with each occurrence of VALUE replaced by *C*, is not false for any row of *T*.

A domain constraint is satisfied by the result of a <cast specification> if and only if the specified <search condition>, with each occurrence of VALUE replaced by that result, is not false.

4.17.4 Assertions

An assertion is a named constraint that may relate to the content of individual rows of a table, to the entire contents of a table, or to a state required to exist among a number of tables.

An assertion is described by an assertion descriptor. In addition to the components of every constraint descriptor an assertion descriptor includes:

— the <search condition>.

An assertion is satisfied if and only if the specified <search condition> is not false.

4.18 Functional dependencies and candidate keys

A *functional dependency* is a truth-valued statement pertaining to a table. If *A* and *B* are both arbitrary subsets (possibly empty) of the columns of a table *T*, then:

$A \mapsto B$

means ''*A determines B*'', or ''*B is functionally dependent on A*''. If the statement holds true in *T*, then any two rows of *T* that agree in value in every column in *A* also agree in value in every column in *B*. The set of columns *A* is said to be a *determinant of* the set of columns *B* and the members of *B* are said to be *dependants of A*.

Associated with any table *T* is a non-empty set of functional dependencies that are constrained to hold true over time, for any value of *T*. For example, if *PK* is the set of columns comprising the defined primary key of some base table *BT*, and *CT* is the set comprising all the columns of *BT*, then the functional dependency *PK* $\mapsto$ *CT* holds true over time in *BT*. The set of functional dependencies is never empty, because, if *CT* is the set comprising all the columns of some table *T*, then *CT* $\mapsto$ *CT* always holds true, trivially, in *T*.

If the functional dependency *CK* $\mapsto$ *CT* holds true in some table *T*, where *CT* comprises all the columns of *T*, and there is no proper subset *CK1* of *CK* such that *CK1* $\mapsto$ *CT* holds true in *T*, then *CK* is a *candidate key* of *T*. Note that if no proper subset of *CT* is a candidate key, then *CT* is the only candidate key of *T*.

This Subclause defines functional dependency and candidate key and specifies a minimal set of rules that a conforming implementation must follow to determine functional dependencies and candidate keys in base tables and <query expression>s.

The rules in this Subclause, though sufficient for most practical purposes, do not guarantee always to give minimal candidate keys. For this reason, they may be freely augmented by implementation-defined rules, where indicated in this Subclause.

This Subclause also specifies a rule whereby one candidate key is noted as the preferred candidate key of any <query expression>. In some cases, this is also designated as the *primary key*.

Other Clauses of this international standard may specify rules in connection with the functional dependencies, candidate keys, preferred candidate keys and primary keys specified by this Subclause.

4.18.1 General rules and definitions

Let *T* be any table. Let *CT* be the set comprising all the columns of *T*, and let *A* and *B* be arbitrary subsets of *CT*, not necessarily disjoint and possibly empty.

Let *A* $\mapsto$ *B* denote the functional dependency of *B* on *A* in *T* holding true if, for any possible value of *T*, any two rows that agree in value for every column in *A* also agree in value for every column in *B*. Two rows agree in value for a column if the two values either are both null or compare as equal under the General Rules of Subclause 8.2, "<comparison predicate>". Let the phrase ''*A* $\mapsto$ *B* is a functional dependency in *T*'' denote that *A* $\mapsto$ *B* holds true for any possible value of *T*.

If *X* $\mapsto$ *Y* is some functional dependency in some table *T*, then *X* is a determinant in *T*.

Let *C1* $\mapsto$ *B* denote that *B* is functionally dependent on the set containing exactly one column, *C1*. Let *A* $\mapsto$ *C2*, for example, denote that the single column *C2* is functionally dependent on the set of columns *A*.

Let *A* $\mapsto$ *B* and *C* $\mapsto$ *D* be any two functional dependencies in *T*. The following are also functional dependencies in *T*:

— *A* UNION (*C* DIFFERENCE *B*) $\mapsto$ *B* UNION *D*

— *C* UNION (*A* DIFFERENCE *D*) $\mapsto$ *B* UNION *D*

The term *candidate key* is defined as follows:

> If *K1* $\mapsto$ *CT* is a functional dependency in *T* and there is no subset *K2* of *K1* such that *K2* $\mapsto$ *CT* is also a functional dependency in *T*, then *K1* is a candidate key of *T*.

Every table has an associated non-empty set of functional dependencies and a non-empty set of candidate keys.

The set of functional dependencies is non-empty because $X \mapsto X$ for any X. A functional dependency of this form is an axiomatic functional dependency, as is $X \mapsto Y$ where Y is a subset of X. $X \mapsto Y$ is a non-axiomatic functional dependency if Y is not a subset of X.

The set of candidate keys is non-empty because, if no proper subset of *CT* is a candidate key, then *CT* is a candidate key.

Let Φ denote the empty set.

Let *BPK-member* denote a property of a column, such that ''*C1* is *BPK-member*'' and ''*C2* is not *BPK-member*'' can be written. A column *C3* is *BPK-member* if it is a unique column of some unique constraint *UC* of some table, and *UC* specifies PRIMARY KEY; otherwise *C3* is not *BPK-member*.

Let a column *C1* be a *counterpart* of a column *C2* if *C1* is specified by a <column reference> (or by a <value expression> that is a <column reference>) that references *C2*. If *C2* is a counterpart of *C1* and *C3* is a counterpart of *C2*, then *C3* is a counterpart of *C1*.

4.18.2 Functional dependencies in a base table

Let *T* be a base table and let *CT* be the set comprising all the columns of *T*.

A column *C1* of *T* is or is not *BPK-member*, in accordance with the General Rules of Subclause 4.18.1, "General rules and definitions".

If the <table definition> of *T* includes a PRIMARY KEY specification, then let *UCL* be the <unique column list> of the primary key; otherwise let *UCL* be *CT*.

$UCL \mapsto CT$ is a functional dependency in T.

Implementation-defined rules may determine other functional dependencies in *T*.

4.18.3 Functional dependencies in <table value constructor>

Let *R* be the result of a <table value expression>, and let *CR* be the set comprising all the columns of *R*.

No column of *R* is *BPK-member*.

If *R* directly contains exactly one <row value constructor>, then $\Phi \mapsto CR$ is a functional dependency in *R*; otherwise $CR \mapsto CR$ is the only functional dependency in *R*.

4.18.4 Functional dependencies in a <joined table>

Let *T1* and *T2* denote the tables identified by the first and second <table reference>s of some <joined table> *JT*. Let *R* denote the table that is the result of *JT*.

Every column of *R* has some counterpart in either *T1* or *T2*. If NATURAL is specified or the <join specification> is a <named columns join>, then some columns of *R* may have counterparts in both *T1* and *T2*.

A column *C1* of *R* is *BPK-member* if *C1* has some counterpart in *T1* or *T2* that is *BPK-member*; otherwise *C1* is not *BPK-member*.

Case:

— If the <join type> is UNION, then no non-axiomatic functional dependency in *T1* or *T2* is a functional dependency in *R*.

— If the <join type> is FULL, then no non-axiomatic functional dependency in *T1* or *T2* whose determinant includes any column that is not *BPK-member* is a functional dependency in *R*.

— If the <join type> is LEFT, then no non-axiomatic functional dependency in *T2* whose determinant includes any column that is not *BPK-member* is a functional dependency in *R*.

— If the <join type> is RIGHT, then no non-axiomatic functional dependency in *T1* whose determinant includes any column that is not *BPK-member* is a functional dependency in *R*.

Except for those discarded under the previous rule, all functional dependencies in *T1* and *T2* are functional dependencies in *R*.

Case:

— If a <cross join> is specified, then there are no further functional dependencies in *R*.

— If a <join condition> is specified and *SC* is the <search condition> directly contained in that <join condition>, then any functional dependencies in the <query expression>

SELECT * FROM *T1*, *T2* WHERE *SC*

are also functional dependencies in *R*.

— Otherwise, further functional dependencies in *R* are determined by treatment of equivalent <query expression>s given in Subclause 7.8, "<joined table>", always taking the <join type> as INNER (even if an <outer join type> is actually given).

4.18.5 Functional dependencies in the result of a <from clause>

Let *R* be the result of some <from clause> *FC*.

If *T* is a <table reference> directly contained in *FC*, then all functional dependencies in *T* are functional dependencies in *R*.

4.18.6 Functional dependencies in the result of a <where clause>

Let *T* be the table that is the operand of the <where clause>. Let *R* be the result of the <where clause>.

Every column in *R* has a counterpart in *T*. A column *C1* in *R* is *BPK-member* if its counterpart in *T* is *BPK-member*; otherwise *C1* is not *BPK-member*.

All functional dependencies in *T* are functional dependencies in *R*.

Let *T1* be *T*, extended to the right with columns arising from <value expression>s contained in the <search condition>, as follows:

Case:

— A <value expression> that is a <literal>, a <general value specification>, or a <datetime value function> is a constant column in *T1*.

— A <value expression> *VE* that is not a <column reference> specifies a computed column *CC* in *T1*. For every row in *T1*, the value in *CC* is the result of *VE*.

If *CONST* is some constant column in *T1*, then $\Phi \mapsto CONST$ is a functional dependency in *T1*.

If *CC* is some computed column in *T1*, then let *OP1, OP2, . . .* be the operands of the <value expression> *VE* that specifies *CC*.

{*OP1, OP2, . . .* } ↦ *CC* is a functional dependency in *T1*.

Further functional dependencies may be determined by implementation-defined rules.

If the <search condition> constrains some column *C1* of *T1* to be equal to some other column *C2* of *T1*, then *C1* ↦ *C2* and *C2* ↦ *C1* are functional dependencies in *T1*.

Note: A conforming implementation is required minimally to detect equality constraints implied by simple sub-expressions of the form *C1* = *C2*, not negated and not unconstrained by the use of ''OR'', where *C1* and *C2* are columns of *T1*.

4.18.7 Functional dependencies in the result of a <group by clause>

Let *T1* be the table that is the operand of the <group by clause>, and let *R* be the result of the <group by clause>.

Let *G* be the set of columns specified by the <collated column reference list> of the <group by clause>.

Note: A <query specification> that does not directly contain a <group by clause>, but whose directly contained <select list> includes a <set function specification>, is treated by this Subclause, with the empty set as *G*.

The columns of *R* are the columns of *G*, with an additional column *CI*, whose value in any particular row of *R* somehow denotes the subset of rows of *T1* that is associated with the combined value of the columns of *G* in that row.

Any column *GC* of *G* has a counterpart in *T1*. *GC* is *BPK-member* if its counterpart in *T1* is *BPK-member*; otherwise, *GC* is not *BPK-member*.

G ↦ *CI* is a functional dependency in *R*.

Note: Any <set function specification> that is specified in conjunction with *R* is necessarily a function of *CI*. If *SFVC* denotes the column containing the results of such a <set function specification>, then *CI* ↦ *SFVC* holds true, and it follows that *G* ↦ *SFVC* is a functional dependency in the table containing *SFVC*.

4.18.8 Functional dependencies in the result of a <having clause>

Let *T1* be the table that is the operand of the <having clause>, let *SC* be the <search condition> directly contained in the <having clause>, and let *R* be the result of the <having clause>.

Any functional dependency in the <query expression>

SELECT * FROM *T1* WHERE *SC*

is a functional dependency in *R*.

4.18.9 Functional dependencies in a <query specification>

Let *T* be the <table expression> directly contained in the <query specification> and let *R* be the result of the <query specification>.

Let *SL* be the <select list> of the <query specification>.

Let *T1* be *T* extended to the right with columns arising from <value expression>s contained in the <select list>, as follows.

Case:

— A <value expression> that is a <literal>, a <general value specification>, or a <datetime value function> is a constant column in *T1*.

— A <value expression> *VE* that is not a <column reference> specifies a computed column *CC* in *T1*. For every row in *T1*, the value in *CC* is the result of *VE*.

Let *dcr* be a column of *R* that arises from the use of * in *SL* or by the specification of a <column reference> as a <value expression> directly contained in *SL*. *DCR* has counterparts in *T* and *T1*. If the counterpart of *DCR* in *T* is *BPK-member*, then *dcr* is *BPK-member*; otherwise *dcr* is not *BPK-member*.

If $A \mapsto B$ is some functional dependency in *T*, then $A \mapsto B$ is a functional dependency in *T1*.

Let *CC* be the column specified by some <value expression> *VE* in the <select list>. Any functional dependency in the <query expression>

SELECT * FROM *T* WHERE *VE* = *VE*

is a functional dependency in *T1*.

Furthermore, if *OP1, OP2, . . .* are the operands of *VE*, then $\{OP1, OP2, \ldots\} \mapsto CC$ is a functional dependency in *T1*.

Let $C \mapsto D$ be some functional dependency in *T1*. If all the columns of *C* have counterparts in *R*, then let *DR* be the set comprising those columns of *D* that have counterparts in *R*. $C \mapsto DR$ is a functional dependency in *R*.

4.18.10 Functional dependencies in a <query expression>

A <query expression> that is a <query term> that is a <query primary> that is a <simple table> or a <joined table> is covered by previous Subclauses of this Clause.

If the <query expression> specifies UNION, EXCEPT or INTERSECT, then let *T1* and *T2* be the left and right operand tables and let *R* be the result. Let *CR* be the set comprising all the columns of *R*.

Each column of *R* has a counterpart in *T1* and a counterpart in *T2*.

Case:

— If EXCEPT is specified, then a column *CR* of *R* is *BPK-member* if its counterpart in *T1* is *BPK-member*; otherwise, *CR* is not *BPK-member*.

— If UNION is specified, then a column *CR* of *R* is *BPK-member* if its counterparts in *T1* and *T2* are both *BPK-member*; otherwise *CR* is not *BPK-member*.

— If INTERSECT is specified, then a column *CR* of *R* is *BPK-member* if either of its counterparts in *T1* and *T2* is *BPK-member*; otherwise *CR* is not *BPK-member*.

Case:

— If UNION is specified, then no non-axiomatic functional dependency in *T1* or *T2* is a functional dependency in *R*, apart from any functional dependencies determined by implementation-defined rules.

— If EXCEPT is specified, then all functional dependencies in *T1* are functional dependencies in *R*.

— If INTERSECT is specified, then all functional dependencies in *T1* and all functional dependencies in *T2* are functional dependencies in *R*.

Note: Other functional dependencies may be determined according to implementation-defined rules.

4.18.11 Functional dependencies in a <recursive union>

To Be Provided!

4.18.12 Selection of primary key or preferred candidate key

Let *CK* be the set of candidate keys of a table *T*.

Note: Because a candidate key is a set (of columns), *CK* is therefore a set of sets (of columns).

Let *PCK* be the set: *P* such that *P* is a member of *CK* and every member (column) of *P* is *BPK-member*.

Case:

— If *PCK* is not empty, then some member of *PCK* is the primary key of *T*. If *PCK* has more than one member, then the primary key is chosen according to the ''left-most'' rule, below.

— Otherwise, some member of *CK* is the preferred candidate key of *T*. If *CK* has more than one member, then the preferred candidate key is chosen according to the ''left-most'' rule, below.

— The ''left-most'' rule:

This rule uses the ordering of the columns of a table, as specified elsewhere in this standard.

To determine the left-most of two sets of columns of *T*, first list each set in the order of the column-numbers of its members, extending the shorter list with zeros to the length of the longer list. Then, starting at the left of each ordered list, step forward until a pair of unequal column numbers, one from the same position in each list, is found. The list containing the number that is the smaller member of this pair identifies the left-most of the two sets of columns of *T*.

To determine the left-most of more than two sets of columns of *T*, take the left-most of any two sets, then pair that with one of the remaining sets and take the left-most, and so on until there are no remaining sets.

4.19 Triggers

****Editor's Note****

I have added a placeholder for some Concepts about triggers, which seems rather desirable. The prime purpose is to make the existing Subclause into a sub-sub-clause called "Triggered actions" as 4.x.1 so that the new sub-sub-clause proposed by X3H2-92-210/CBR-054, part 1 can be 4.x.2, thus not violating the ISO and ANSI prohibitions against having a .1 if there's not at least a .2. Of course, you are all welcome to propose text for this new Subclause 4.19, "Triggers"!

A trigger is defined by a <trigger definition>. A <trigger definition> specifies a trigger that is described by a trigger descriptor . A trigger descriptor includes:

— The name of the trigger;

— The name of the table that is the subject table of the trigger;

— The trigger action time (BEFORE or AFTER) of the trigger;

— The trigger event (INSERT, DELETE, or UPDATE) of the trigger;

— The old values correlation name, if any, of the trigger;

— The new values correlation name, if any, of the trigger;

— All of the triggered actions of the trigger;

— If the trigger event is UPDATE, then the trigger column list for the trigger event of the trigger, as well as an indication of whether the trigger column list was explicit or implicit;

4.19.1 Triggered actions

A schema may include the definition of SQL-statements that are to be executed before or after a row is inserted into or deleted from a table or executed during the update of a row of a table. The execution of such a triggered action resulting from the insertion, deletion, or updating of a row in a table may cause the triggering of further triggered actions in other affected tables.

4.19.2 Execution of triggered actions

The execution of triggered actions depends on the cursor mode of the current SQL-transaction. If the cursor mode is set to *cascade off*, then the execution of the <triggered SQL statement>s is effectively deferred until enacted implicitly be execution of a <commit statement> or a <close statement>. Otherwise, the <triggered SQL statement>s are effectively executed either before or after the execution of each SQL-statement, as determined by the specified <trigger action time>.

When the <triggered SQL statement>s are executed other than at the end of an SQL-statement, if their execution is not successful, then an exception condition is raised and the SQL-statement that caused the trigger to be executed has no effect other than entering exception information into the diagnostics area. When a <commit statement> implicitly causes a <triggered SQL statement> to be executed and the execution of the <triggered SQL statement> is not successful, then an exception condition is raised and the transaction is terminated by an implicit <rollback statement>.

4.20 SQL-schemas

An SQL-schema is a persistent descriptor that includes:

— the <schema name> of the SQL-schema;

— the <authorization identifier> of the owner of the SQL-schema;

— The <character set name> of the default character set for the SQL-schema; and

— the descriptor of every component of the SQL-schema.

In this
ANSI American
ISO International
standard, the term ‘‘schema’’ is used only in the sense of SQL-schema. Each component descriptor is either a domain descriptor, a base table descriptor, a view descriptor, an assertion descriptor, a privilege descriptor, a character set descriptor, a collation descriptor, a translation descriptor, an abstract data type descriptor, a type template descriptor, or a
ANSI routine
ISO function
descriptor. The persistent objects described by the descriptors are said to be *owned by* or to have been *created by* the <authorization identifier> of the schema.

A schema is created initially using a <schema definition> and may be subsequently modified incrementally over time by the execution of <SQL schema statement>s. <schema name>s are unique within a catalog.

A <schema name> is explicitly or implicitly qualified by a <catalog name> that identifies a catalog.

Base tables and views are identified by <table name>s. A <table name> consists of a <schema name> and an <identifier>. For a persistent table, the <schema name> identifies the schema in which the base table or view identified by the <table name> was defined. Base tables and views defined in different schemas can have <identifier>s that are equal according to the General Rules of Subclause 8.2, "<comparison predicate>".

If a reference to a <table name> does not explicitly contain a <schema name>, then a specific <schema name> is implied. The particular <schema name> associated with such a <table name> depends on the context in which the <table name> appears and is governed by the rules for <schema qualified name>. The default schema for <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s is initially implementation-defined but may be changed by the use of <set schema statement>s.

4.21 Catalogs

Catalogs are named collections of schemas in an SQL-environment. An SQL-environment contains zero or more catalogs. A catalog contains one or more schemas, but always contains a schema named INFORMATION_SCHEMA that contains the views and domains of the Information Schema. The method of creation and destruction of catalogs is implementation-defined. The set of catalogs that can be referenced in any SQL-statement, during any particular SQL-transaction, or during the course of an SQL-session is also implementation-defined. The default catalog for a <module> whose <module authorization clause> does not specify an explicit <catalog name> to qualify the <schema name> is implementation-defined. The default catalog for <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s is initially implementation-defined but may be changed by the use of <set catalog statement>s.

4.22 Clusters of catalogs

A cluster is an implementation-defined collection of catalogs. Exactly one cluster is associated with an SQL-session and it defines the totality of the SQL-data that is available to that SQL-session.

An instance of a cluster is described by an instance of a *definition schema*. Given some SQL-data object, such as a view, a constraint, a domain, or a base table, the descriptor of that object, and of all the objects that it directly or indirectly references, are in the same cluster of catalogs. For example, no referential constraint and no <query expression> can reference any SQL-data object whose descriptors are not in the same cluster (that is, they cannot ‘‘cross’’ a cluster boundary).

Whether or not any catalog can occur simultaneously in more than one cluster is implementation-defined.

Within a cluster, no two catalogs have the same name.

4.23 SQL-data

SQL-data is any data described by schemas that is under the control of an SQL-implementation in an SQL-environment.

4.24 SQL-environment

An *SQL-environment* comprises the following:

— an SQL-implementation capable of processing some Level (Entry SQL, Intermediate SQL, or Full SQL) or the entirety of this
ANSI American
ISO International
Standard and at least one binding style; see Clause 25, "Conformance" for further information about binding styles;

> ****Editor’s Note****
> The previous bullet refers to the Levels of the language, but fails to mention the highest Level for SQL3. I cannot fill this in editorially because that Level has no official name. (I do *not* consider "Overfull" an appropriate name!) Instead, I added "or the entirety" as suggested by Amelia Carlson.

— zero or more catalogs;

— zero or more <authorization identifier>s;

— zero or more <module>s; and

— the SQL-data described by the schemas in the catalogs.

An SQL-environment may have other implementation-defined contents.

The rules determining which <module>s are considered to be within an SQL-environment are implementation-defined.

4.25 Modules

A <module> is an object specified in the module language. A <module> is either a persistent <module> or an SQL-session <module>. <module>s are created and destroyed by the <create module statement> and <drop module statement>, respectively. They may also be created and destroyed by implementation-defined mechanisms (which can include the granting and revoking of module privileges). A <module> consists of a <module name>, a <language clause>, a <module authorization clause> with either or both of a <module authorization identifier> and a <schema name>, an optional <module character set specification> that identifies the character repertoire used for expressing the names of schema objects used in the <module>, zero or more <temporary table declaration>s, zero or more cursors specified by <declare cursor>s, and one or more <routine>s. All <identifier>s contained in the <module> are expressed in either <SQL language character> or the character repertoire indicated by <module character set specification> unless they are specified with ''<introducer>''.

A compilation unit is a segment of executable code, possibly consisting of one or more subprograms. A <module> is associated with a compilation unit during its execution. A single <module> may be associated with multiple compilation units and multiple <module>s may be associated with a single compilation unit. The manner in which this association is specified, including the possible requirement for execution of some implementation-defined statement, is implementation-defined. Whether a compilation unit may invoke or transfer control to other compilation units, written in the same or a different programming language, is implementation-defined.

A <module> is described by a module descriptor. A module descriptor contains:

- the name of the <module>,
- the descriptor of the character set used for representing the <module>,
- the schema name used for implicit qualification of unqualified names in the <module>
- the <module authorization identifier>,
- the <language clause> of the <module>,
- the text of the <module definition>

4.26 Routines

ANSI Only–SQL3

A <routine> is a subprogram. A <routine> consists of a <routine name>, a sequence of <parameter declarations>s, and a <routine body>.

A <routine> is either a component of a <module>, a component of a <compound statement>, or an object in a <schema>.

A <routine> is either a function, which returns a value, or a procedure, which does not return a value.

4.26 Routines

A <routine> is either an SQL-invoked <routine> or an externally-invoked <routine>, depending on whether the <caller language clause> of the <routine> specifies ''SQL'' or some standard programming language. An SQL-invoked <routine> (function or procedure) is invoked from within SQL. An externally-invoked <routine> (function or procedure) is invoked by ''call'' statements in compilation units of the specified standard programming language.

A <routine> is either an SQL <routine> or an external <routine>. An SQL <routine> is a <routine> whose <routine body> is an <SQL statement> (which can be a <compound statement>). An external <routine> is a <routine> whose <routine body> is an <external body reference> that identifies a routine body provided externally as a program written in a standard programming language. An SQL-invoked <routine> may be either an SQL <routine> or an external <routine>. An externally-invoked <routine> must be an SQL <routine>.

An invocation of a <routine> specifies the <routine name> of the <routine> and supplies a sequence of argument values corresponding to the <parameter declaration>s of the <routine>. Such invocation causes the <routine body> to be executed.

ISO Only—caused by ANSI changes not yet considered by ISO

A <routine> consists of a <routine name>, a sequence of <parameter declaration>, and a <routine body>.

A <routine body> is either an <SQL procedure statement> (which could be a <compound statement>), or an <external body reference> that identifies a routine body that is provided externally as a program written in a standard programming language.

A <routine> is either a component of a <module> or an object in a <schema>.

The <caller language clause> of a <routine> specifies whether it will be invoked from within SQL, by <routine invocation>s or <call statement>s, or by a compilation unit of a specified standard programming language, by means of "call" statements or function invocations.

An invocation of a <routine> specifies the <routine name> of the <routine> and supplies a sequence of argument values corresponding in number and in <data type> to the <parameter declaration>s of the <routine>. Such invocation causes the <SQL procedure statement> or external routine body to be executed.

A <routine> is either a function, which returns a value, or a procedure, which does not return a value. Function invocations can be specified in <value expression>s and <search condition>s in both schemas and modules. Procedure invocations are specified in <call statement>s or in compilation units.

Different <routine>s can have the same <routine name>. This is referred to an *overloading* the <routine name>. Overloading is allowed only for <routine>s that are called from SQL. The parameter declaration lists of such routines must be sufficiently different to distinguish which of the routines is to be invoked for a given <routine invocation>.

A routine is described by a routine descriptor. A routine descriptor contains:

— The <routine name>;

ANSI Only–SQL3

— If the routine is an SQL-invoked routine, then the specific name of the routine;

— The name, data type, position, and the indication of whether the parameter is IN, OUT, or INOUT of each of the routine's parameters.

ISO Only—caused by ANSI changes not yet considered by ISO

— The name, data type, and position of each of the routine's parameters;

— If the routine is a function, then the data type returned by the function; and

— If the routine is defined within an abstract data type definition or a table definition, then an indicator of its encapsulation level.

ANSI Only–SQL3

— If the routine is an SQL-invoked <routine> that is an SQL <routine>, then the function type of the routine;

— The name of the caller language of the routine;

— If the routine is an external routine, then the external name of the routine;

— If the routine is an external routine, then the name of the language in which the body of the routine is written;

— If the routine is an external routine, then an indication of whether the routine is a variant routine or not;

4.27 Parameters

ANSI Only–SQL3

A parameter is declared in a <routine> by a <parameter declaration>. A parameter either assumes or supplies the value of the corresponding argument in the call of the <routine>.

A parameter is either an SQL-supplied parameter or an externally-supplied parameter, depending on whether it is declared in an SQL-invoked or externally-invoked <routine>.

A <parameter declaration> specifies the <data type> of its value. The <data type>s of externally supplied parameters map to the host language types of the arguments.

SQL supplied parameters can be null. Externally supplied parameters cannot be null, except through the use of indicator parameters.

ISO Only—caused by ANSI changes not yet considered by ISO

A parameter is declared in a <routine> by a <parameter declaration>. The <parameter declaration> specifies the <data type> of its value. A parameter either assumes or supplies the value of the corresponding argument in the call of that <routine>. These <data type>s map to host language types and are not nullable except through the use of additional indicator variables.

4.27.1 Status parameters

The SQLSTATE and SQLCODE parameters are status parameters. They are set to status codes that indicate either that a call of the <routine> completed successfully or that an exception condition was raised during execution of the <routine>.

Note: The SQLSTATE parameter is the preferred status parameter. The SQLCODE parameter is a deprecated feature that is supported for compatibility with earlier versions of this
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

A <routine> must specify either the SQLSTATE parameter or the SQLCODE parameter or both. The SQLSTATE parameter is a character string parameter for which exception values are defined in Clause 24, "Status codes". The SQLCODE parameter is an integer parameter for which the negative exception values are implementation-defined.

If a condition is raised that causes a statement to have no effect other than that associated with raising the condition (that is, not a completion condition), then the condition is said to be an *exception condition* or *exception*. If a condition is raised that permits a statement to have an effect other than that associated with raising the condition (corresponding to an SQLSTATE class value of *successful completion*, *warning*, or *no data*), then the condition is said to be a *completion condition*.

Exception conditions or completion conditions may be raised during the execution of an <SQL procedure statement>. If the <SQL procedure statement> is not a <compound statement>, then one of the exception conditions becomes the active condition when the <SQL procedure statement> terminates. If the active condition is an exception condition, then it will be called the active exception condition. If the active condition is a completion condition, then it will be called the active completion condition. If the <SQL procedure statement> is a <compound statement>, then the active condition is dependent on the exception handler specified in the <compound statement>.

4.27.2 Data parameters

A data parameter is a parameter that is used to either assume or supply the value of data exchanged between a host program and an SQL-implementation.

4.27.3 Indicator parameters

An indicator parameter is an integer parameter that is specified immediately following another parameter. Its primary use is to indicate whether the value that the other parameter assumes or supplies is a null value and the appropriate null state of that null value, if any. An indicator parameter cannot immediately follow another indicator parameter.

The other use for indicator parameters is to indicate whether string data truncation occurred during a transfer between a host program and an SQL-implementation in parameters or host variables. If a non-null character string value is transferred and the length of the target data item is sufficient to accept the entire source data item, then the indicator parameter or variable is set to 0 to indicate that truncation did not occur. However, if the length of the target data item is insufficient, the indicator parameter or variable is set to the length (in characters or bits, as appropriate) of the source data item to indicate that truncation occurred and to indicate original length in characters or bits, as appropriate, of the source.

4.28 Diagnostics area

The diagnostics area is a place where completion and exception condition information is stored when an SQL-statement is executed. There is one diagnostics area associated with an SQL-agent, regardless of the number of <module>s that the SQL-agent includes or the number of connections in use.

At the beginning of the execution of any <routine> that contains an <SQL procedure statement> that is not an <SQL diagnostics statement>, the diagnostics area is emptied. An implementation must place information about a completion condition or an exception condition reported by SQLCODE or SQLSTATE into this area. If other conditions are raised, an implementation may place information about them into this area. The diagnostics area is emptied during the execution of a <signal statement>. Information is added to the diagnostics area during the execution of a <resignal statement>.

<routine>s containing <SQL diagnostics statement>s return a code indicating completion or exception conditions for that statement via SQLCODE or SQLSTATE, but do not modify the diagnostics area.

An SQL-agent may choose the size of the diagnostics area with the <set transaction statement>; if an SQL-agent does not specify the size of the diagnostics area, then the size of the diagnostics area is implementation-dependent, but shall always be able to hold information about at least one condition. An implementation may place information into this area about fewer conditions than are specified. The ordering of the information about conditions placed into the diagnostics area is implementation-dependent, except that the first condition in the diagnostics area always corresponds to the condition specified by the SQLSTATE or SQLCODE value.

4.29 Standard programming languages

This
ANSI American
ISO International
Standard specifies the actions of <routine>s in <module>s when those <routine>s are called by programs that conform to certain specified programming language standards. The term ‘‘standard *PLN* program’’, where *PLN* is the name of a programming language, refers to a program that conforms to the standard for that programming language as specified in Clause 2, "Normative

references". This
ANSI American
ISO International
Standard also specifies a mechanism whereby SQL language may be embedded in programs that otherwise conform to any of the same specified programming language standards.

Note: In this
ANSI American
ISO International
Standard, for the purposes of interfacing with programming languages, the data types DATE, TIME, TIMESTAMP, and INTERVAL must be converted to or from character strings in those programming languages by means of a <cast specification>. It is anticipated that future evolution of programming language standards will support data types corresponding to these four SQL data types; this standard will then be amended to reflect the availability of those corresponding data types. The data type CHARACTER is also mapped to character strings in the programming languages. However, because the facilities available in the programming languages do not provide the same capabilities as those available in SQL, there must be agreement between the host program and SQL regarding the specific format of the character data being exchanged. Specific syntax for this agreement is provided in thie
ANSI American
ISO International
standard. For standard programming languages, C, COBOL, Fortran, and Pascal, bit strings are mapped to character variables in the host language in a manner described in Subclause 21.1, "<embedded SQL host program>". For standard programming languages Ada and PL/I, bit string variables are directly supported.

****Editor's Note****
Standard programming language MUMPS is not mentioned herein. It should be resolved in some way. Do we want to define handling of BITs in MUMPS?

4.30 Cursors

A cursor is specified by a <declare cursor>, <dynamic declare cursor>, or <allocate cursor statement>.

For every <declare cursor> or <dynamic declare cursor> in a <module>, a cursor is effectively created when an SQL-transaction (see Subclause 4.37, "SQL-transactions") referencing the <module> is initiated, and destroyed when that SQL-transaction is terminated.

ANSI Only–SQL3

For every <declare cursor> in a <compound statement>, a cursor is effectively created each time the <compound statement> is executed, and destroyed when that execution completes.

A cursor is also effectively created when an <allocate cursor statement> is executed within a SQL-transaction and destroyed when that SQL-transaction is terminated. In addition, an extended dynamic cursor is destroyed when a <deallocate prepared statement> is executed that deallocates the prepared statement on which the extended dynamic cursor is based.

A cursor is in either the open state or the closed state. The initial state of a cursor is the closed state. A cursor is placed in the open state by an <open statement> or <dynamic open statement> and returned to the closed state by a <close statement> or <dynamic close statement>, a <commit statement>, or a <rollback statement>.

A cursor in the open state identifies a table, an ordering of the rows of that table, and a position relative to that ordering. If the <declare cursor> does not include an <order by clause>, or includes an <order by clause> that does not specify the order of the rows completely, then the rows of the table have an order that is defined only to the extent that the <order by clause> specifies an order and is otherwise implementation-dependent.

When the ordering of a cursor is not defined by an <order by clause>, the relative positions of two rows is implementation-dependent. When the ordering of a cursor is partially determined by an <order by clause>, then the relative positions of two rows are determined only by the <order by clause>; if the two rows have equal values for the purpose of evaluating the <order by clause>, then their relative positions are implementation-dependent.

A cursor is either *read-only* or *updatable*. If the table identified by a cursor is not updatable or if INSENSITIVE is specified for the cursor, then the cursor is read-only; otherwise, the cursor is updatable. The operations of update and delete are not allowed for read-only cursors.

The position of a cursor in the open state is either before a certain row, on a certain row, or after the last row. If a cursor is on a row, then that row is the current row of the cursor. A cursor may be before the first row or after the last row of a table even though the table is empty. When a cursor is initially opened, the position of the cursor is before the first row.

A <fetch statement> or <dynamic fetch statement> positions an open cursor on a specified row of the cursor's ordering and retrieves the values of the columns of that row. An <update statement: positioned> or <dynamic update statement: positioned> updates the current row of the cursor. A <delete statement: positioned> or <dynamic delete statement: positioned> deletes the current row of the cursor.

If an error occurs during the execution of an SQL-statement that identifies an open cursor, then, except where otherwise explicitly defined, the effect, if any, on the position or state of that cursor is implementation-dependent.

The following paragraphs define several terms used to discuss issues relating to cursor sensitivity:

A *DB-change* is any change to SQL-data resulting from the execution of an <insert statement>, an <update statement: positioned>, an <update statement: searched>, a <delete statement: positioned> or a <delete statement: searched>.

A DB-change is said to be *independent* of a cursor *CR* if and only if it is not made by an <update statement: positioned> or a <delete statement: positioned> that is positioned on *CR*.

A DB-change is said to be *significant* to *CR* if and only if it is independent of *CR*, and, had it occurred before *CR* was opened, would have caused the table associated with the cursor to be different in any respect.

If *CR* denotes an open cursor, then let **proc-VC** be the procedure:

— Let *P* be the position of *CR* (either before the first row, after the last row, or on or before some row *R* of the table associated with *CR*).

— *CR* is closed.

— The DB-change is made, including any relevant referential actions.

— *CR* is reopened by applying the General Rules of Subclause 15.2, "<open statement>", using the same *S* that resulting from applying those General Rules in the course of executing the most recent <open statement> or <dynamic open statement> for *CR*.

— If the rows that constitute *CR* or the order of those rows has changed as a result of closing and reopening *CR*, then the position of *CR* is undefined. Otherwise, the position of *CR* is set to *P*.

A change is said to be *invisible* to *CR* if and only if it has no effect on *CR*.

A change is said to be *visible* to *CR* if and only if its effect on *CR* is as if the change had been effected by executing the procedure **proc-VC** on *CR*.

If a cursor is open, and the current SQL-transaction makes a change to SQL-data other than through that cursor, and the <declare cursor> of that cursor did not specify either SENSITIVE or INSENSITIVE, then whether any effect of that change will be visible through that cursor before it is closed is implementation-dependent.

If a cursor is open, and the current SQL-transaction makes a change to SQL-data other than through that cursor and the <declare cursor> for that cursor specified INSENSITIVE, then the effect of that change will not be visible through that cursor before it is closed. Otherwise, whether the effect of such a change will be visible through that cursor before it is closed is implementation-dependent.

4.31 SQL-statements

Editor's Note
This Subclause has not been properly updated for the various SQL3 statements and SQL3 work that has been done. See Possible Problem **261**.

4.31.1 Classes of SQL-statements

An SQL-statement is a string of characters that conforms to the Format and Syntax Rules specified in this international standard. Most SQL-statements can be prepared for execution and executed in one of a number of ways. These are:

— in a module, in which case it is prepared when the module is created (see Subclause 4.25, "Modules") and executed when the containing procedure is called.

— in an embedded SQL host program, in which case it is prepared when the embedded SQL host program is preprocessed (see Subclause 4.32, "Embedded syntax").

— being prepared and executed by the use of SQL-dynamic statements (which are themselves executed in one of the foregoing two ways—see Subclause 4.33, "SQL dynamic statements").

— direct invocation, in which case it is effectively prepared immediately prior to execution (see Subclause 4.34, "Direct invocation of SQL").

There are at least five ways of classifying SQL-statements:

— According to their effect on SQL objects, whether persistent objects, *i.e.*, SQL-data and schemas, or transient objects, such as SQL-sessions and other SQL-statements.

— According to whether or not they start a transaction, or can, or must, be executed when no transaction is active.

— According to whether or not they may be embedded.

— According to whether they may be dynamically prepared and executed.

— According to whether or not they may be directly executed.

This
ANSI American
ISO International
Standard permits implementations to provide additional, implementation-defined, statements that may fall into any of these categories. This Subclause will not mention those statements again, as their classification is entirely implementation-defined.

4.31.2 SQL-statements classified by function

The following are the main classes of SQL-statements:

— SQL-schema statements; these may have a persistent effect on schemas

— SQL-data statements; some of these, the SQL-data change statements, may have a persistent effect on SQL-data

— SQL-transaction statements; except for the <commit statement>, these, and the following classes, have no effects that persist when a session is terminated

— SQL-control statements

— SQL-connection statements

— SQL-session statements

— SQL-dynamic statements

— SQL-diagnostics statements

— SQL embedded exception declaration

The following are the SQL-schema statements:

— <schema definition>

— <drop schema statement>

— <domain definition>

— <drop domain statement>

— <table definition>

— <drop table statement>

— <view definition>

— <drop view statement>

— <assertion definition>

— <drop assertion statement>

— <alter table statement>

— <alter domain statement>

— <grant statement>

— <revoke statement>

— <character set definition>

— <drop character set statement>

— <collation definition>

— <drop collation statement>

— <translation definition>

— <drop translation statement>

— <trigger definition>

— <drop trigger statement>

— <null class definition>

— <drop null class statement>

— <routine>

— <drop routine statement>

— <abstract data type definition>

— <type template definition>

— <drop type template statement>

— <drop abstract data type statement>

— <module definition>

— <drop module statement>

— <role definition>

— <grant role statement>

— <revoke role statement>

— <drop role statement>

The following are the SQL-data statements:

— <temporary table declaration>

— <declare cursor>

— <dynamic declare cursor>

— <allocate cursor statement>

— <dynamic select statement>

— <open statement>

— <dynamic open statement>

— <close statement>

— <dynamic close statement>

— <fetch statement>

— <dynamic fetch statement>

— <select statement: single row>

— <direct select statement: multiple rows>

— <dynamic single row select statement>

— All SQL-data change statements

The following are the SQL-data change statements:

— <insert statement>

— <delete statement: searched>

— <delete statement: positioned>

— <dynamic delete statement: positioned>

— <preparable dynamic delete statement: positioned>

— <update statement: searched>

— <update statement: positioned>

— <dynamic update statement: positioned>

— <preparable dynamic update statement: positioned>

The following are the SQL-transaction statements:

— <set transaction statement>

— <set constraints mode statement>

— <commit statement>

— <rollback statement>

The following are the SQL-control statements:

— <call statement>

— <return statement>

— <exception handler>

— <exception declaration>

— <compound statement>

— <case statement>

— <if statement>

— <leave statement>

— <loop statement>

ANSI Only–SQL3

— <for statement>

— <assignment statement>

— <new statement>

— <destroy statement>

• 2 list items deleted

The following are the SQL-connection statements:

— <connect statement>

— <set connection statement>

— <disconnect statement>

The following are the SQL-session statements:

— <set catalog statement>

— <set schema statement>

— <set names statement>

— <set session authorization identifier statement>

— <set local time zone statement>

The following are the SQL-dynamic statements:

— <execute immediate statement>

— <allocate descriptor statement>

— <deallocate descriptor statement>

— <get descriptor statement>

— <set descriptor statement>

— <prepare statement>

— <deallocate prepared statement>

— <describe input statement>

— <describe output statement>

— <execute statement>

The following are the SQL-diagnostics statements:

— <get diagnostics statement>

— <signal statement>

— <resignal statement>

The following is the SQL embedded exception declaration:

— <embedded exception declaration>

4.31.3 Embeddable SQL-statements

The following SQL-statements are embeddable in an embedded SQL host program, and may be the <SQL procedure statement> in a <routine> in a <module>:

— All SQL-schema statements

— All SQL-transaction statements

— All SQL-connection statements

— All SQL-session statements

— All SQL-dynamic statements

— All SQL-diagnostics statements

— The following SQL-data statements:

  • <allocate cursor statement>

  • <open statement>

  • <dynamic open statement>

  • <close statement>

  • <dynamic close statement>

  • <fetch statement>

  • <dynamic fetch statement>

- <select statement: single row>
- <insert statement>
- <delete statement: searched>
- <delete statement: positioned>
- <dynamic delete statement: positioned>
- <update statement: searched>
- <update statement: positioned>
- <dynamic update statement: positioned>

The following SQL-statements are embeddable in an embedded SQL host program, and may occur in a <module>, though not in a <routine>:

— <temporary table declaration>

— <declare cursor>

— <dynamic declare cursor>

The following SQL-statements are embeddable in an embedded SQL host program, but may not occur in a <module>:

— SQL embedded exception declarations

Consequently, the following SQL-data statements are not embeddable in an embedded SQL host program, nor may they occur in a <module>, nor be the <SQL procedure statement> in a <routine> in a <module>:

— <dynamic select statement>

— <dynamic single row select statement>

— <direct select statement: multiple rows>

— <preparable dynamic delete statement: positioned>

— <preparable dynamic update statement: positioned>

4.31.4 Preparable and immediately executable SQL-statements

The following SQL-statements are preparable:

— All SQL-schema statements

— All SQL-transaction statements

— All SQL-session statements

— The following SQL-data statements:

- <delete statement: searched>

- <dynamic select statement>
- <dynamic single row select statement>
- <insert statement>
- <update statement: searched>
- <preparable dynamic delete statement: positioned>
- <preparable dynamic update statement: positioned>
- <preparable implementation-defined statement>

Consequently, the following SQL-statements are not preparable:

— All SQL-connection statements

— All SQL-dynamic statements

— All SQL-diagnostics statements

— SQL embedded exception declarations

— The following SQL-data statements:

- <allocate cursor statement>
- <open statement>
- <dynamic open statement>
- <close statement>
- <dynamic close statement>
- <fetch statement>
- <dynamic fetch statement>
- <select statement: single row>
- <delete statement: positioned>
- <dynamic delete statement: positioned>
- <update statement: positioned>
- <dynamic update statement: positioned>
- <direct select statement: multiple rows>
- <temporary table declaration>
- <declare cursor>
- <dynamic declare cursor>

Any preparable SQL-statement can be executed immediately, with the exception of:

— <dynamic select statement>

— <dynamic single row select statement>

4.31.5 Directly executable SQL-statements

The following SQL-statements may be executed directly:

— All SQL-schema statements

— All SQL-transaction statements

— All SQL-connection statements

— All SQL-session statements

— The following SQL-data statements:

- <temporary table declaration>
- <direct select statement: multiple rows>
- <insert statement>
- <delete statement: searched>
- <update statement: searched>

Consequently, the following SQL-statements may not be executed directly:

— All SQL-dynamic statements

— All SQL-diagnostics statements

— SQL embedded exception declarations

— The following SQL-data statements:

- <declare cursor>
- <dynamic declare cursor>
- <allocate cursor statement>
- <open statement>
- <dynamic open statement>
- <close statement>
- <dynamic close statement>
- <fetch statement>
- <dynamic fetch statement>

- <select statement: single row>
- <dynamic select statement>
- <dynamic single row select statement>
- <delete statement: positioned>
- <dynamic delete statement: positioned>
- <preparable dynamic delete statement: positioned>
- <update statement: positioned>
- <dynamic update statement: positioned>
- <preparable dynamic update statement: positioned>

4.31.6 SQL-statements and transaction states

Whether an <execute immediate statement> starts a transaction depends on what SQL-statement is the value of <SQL statement variable>. Whether an <execute statement> starts a transaction depends on what SQL-statement was the value of <SQL statement variable> when the prepared statement identified by <SQL statement name> was prepared.

Whether a <call statement> starts a transaction not depends upon the <SQL procedure statement> contained in the called <routine>.

Whether a <compound statement> starts a transaction not depends upon the <SQL procedure statement>s contained in the <compound statement>.

Whether an <assignment statement> starts a transaction not depends upon whether the <assignment statement> contains a <subquery>.

The following SQL-statements are transaction initiating SQL-statements, *i.e.*, if there is no current transaction, and a statement of this class is executed, a transaction is initiated:

— All SQL-schema statements

— The following SQL-data statements:

- <allocate cursor statement>
- <dynamic select statement>
- <open statement>
- <dynamic open statement>
- <close statement>
- <dynamic close statement>
- <fetch statement>
- <dynamic fetch statement>
- <select statement: single row>

- <direct select statement: multiple rows>
- <dynamic single row select statement>
- <insert statement>
- <delete statement: searched>
- <delete statement: positioned>
- <dynamic delete statement: positioned>
- <preparable dynamic delete statement: positioned>
- <update statement: searched>
- <update statement: positioned>
- <dynamic update statement: positioned>
- <preparable dynamic update statement: positioned>

— The following SQL-dynamic statements:

- <describe input statement>
- <describe output statement>
- <allocate descriptor statement>
- <deallocate descriptor statement>
- <get descriptor statement>
- <set descriptor statement>
- <prepare statement>
- <deallocate prepared statement>

— <return statement>

The following SQL-statements are not transaction initiating SQL-statements, *i.e.*, if there is no current transaction, and a statement of this class is executed, no transaction is initiated.

— All SQL-transaction statements

— All SQL-connection statements

— All SQL-session statements

— All SQL-diagnostics statements

— SQL embedded exception declarations

— The following SQL-data statements:

- <temporary table declaration>

- <declare cursor>
- <dynamic declare cursor>
- <dynamic select statement>

4.31.7 Asynchronous execution

An <SQL procedure statement> may be executed either synchronously or asynchronously. If an <SQL procedure statement> is executed synchronously, then control is not returned to the SQL-agent until all actions specified for the statement have been performed. If an <SQL procedure statement> is executed asynchronously, then, at the option of the DBMS, control may be returned to the SQL-agent before any of the actions specified for the statement have been performed. Synchronous and asynchronous execution of <SQL procedure statement>s may be intermixed, but the effect of the <SQL procedure statement>s is the same as if they had all been executed synchronously in the order in which they had been initiated. Furthermore, an <SQL procedure statement> initiated for asynchronous execution may be either accepted or rejected by the DBMS. In the latter case, an exception condition is raised: *asynchronous SQL statement not accepted*.

4.32 Embedded syntax

An <embedded SQL host program> (<embedded SQL Ada program>, <embedded SQL C program>, <embedded SQL COBOL program>, <embedded SQL Fortran program>, <embedded SQL MUMPS program>, <embedded SQL Pascal program>, or <embedded SQL PL/I program>) is a compilation unit that consists of programming language text and SQL text. The programming language text shall conform to the requirements of a specific standard programming language. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s, as defined in this
ANSI American
ISO International
Standard. This allows database applications to be expressed in a hybrid form in which SQL-statements are embedded directly in a compilation unit. Such a hybrid compilation unit is defined to be equivalent to a standard compilation unit in which the SQL-statements have been replaced by standard procedure or subroutine calls of SQL <routine>s in a separate SQL <module>, and in which each been removed and the declarations contained therein have been suitably transformed into standard host-language syntax.

An implementation may reserve a portion of the name space in the <embedded SQL host program> for the names of procedures or subroutines that are generated to replace SQL-statements and for program variables and branch labels that may be generated as required to support the calling of these procedures or subroutines; whether this reservation is made is implementation-defined. They may similarly reserve name space for the <module name> and <procedure name>s of the generated <module> that may be associated with the resulting standard compilation unit. The portion of the name space to be so reserved, if any, is implementation-defined.

4.33 SQL dynamic statements

In many cases, the SQL-statement to be executed can be coded into a <module> or into a compilation unit using the embedded syntax. In other cases, the SQL-statement is not known when the program is written and will be generated during program execution. An <execute immediate statement> can be used for a one-time preparation and execution of an SQL-statement. A <prepare statement> is used to prepare the generated SQL-statement for subsequent execution. A <deallocate prepared statement> is used to deallocate SQL-statements that have been prepared with a <prepare statement>. A description of the input parameters for a prepared statement can be obtained by execution of a <describe input statement>. A description of the resultant columns of a <dynamic select statement> or <dynamic single row select statement> can be obtained by execution of a <describe output statement>. For a statement other than a <dynamic select statement>, an <execute statement> is used to associate parameters with the prepared statement and execute it as though it had been coded when the program was written. For a <dynamic select statement>, the prepared <cursor specification> is associated with a cursor via a <dynamic declare cursor> or <allocate cursor statement>. The cursor can be opened and parameters can be associated with the cursor with a <dynamic open statement>. A <dynamic fetch statement> positions an open cursor on a specified row and retrieves the values of the columns of that row. A <dynamic close statement> closes a cursor that was opened with a <dynamic open statement>. A <dynamic delete statement: positioned> is used to delete rows through a dynamic cursor. A <dynamic update statement: positioned> is used to update rows through a dynamic cursor. A <preparable dynamic delete statement: positioned> is used to delete rows through a dynamic cursor when the precise format of the statement isn't known until runtime. A <preparable dynamic update statement: positioned> is used to update rows through a dynamic cursor when the precise format of the statement isn't known until runtime.

The interface for input parameters for a prepared statement and for the resulting values from a <dynamic fetch statement> or the execution of a prepared <dynamic single row select statement> can be either a list of parameters or embedded variables or an SQL descriptor area. An SQL descriptor area consists of zero or more item descriptor areas, together with a COUNT of the number of those item descriptor areas. Each item descriptor area consists of the fields specified in Table 24, "Data types of <key word>s used in SQL item descriptor areas", in Subclause 19.1, "Description of SQL item descriptor areas". The SQL descriptor area is allocated and maintained by the system with the following statements: <allocate descriptor statement>, <deallocate descriptor statement>, <set descriptor statement>, and <get descriptor statement>.

An SQL descriptor area is identified by a <descriptor name>, which is a <simple value specification> whose value is an <identifier>. Two <descriptor name>s identify the same SQL descriptor area if their values, with leading and trailing <space>s removed, are equivalent according to the rules for <identifier> comparisons in Subclause 5.2, "<token> and <separator>".

Dynamic statements can be identified by <statement name>s or by <extended statement name>s. Similarly, dynamic cursors can be identified by <cursor name>s and by <extended cursor name>s. The non-extended names are <identifier>s. The extended names are <target specification>s whose values are <identifier>s used to identify the statement or cursor. Two extended names are equivalent if their values, with leading and trailing <space>s removed, are equivalent according to the rules for <identifier> comparison in Subclause 5.2, "<token> and <separator>".

An SQL descriptor area name may be defined as global or local. Similarly, an extended statement name or extended cursor name may be global or local. The scope of a global name is the SQL-session. The scope of a local name is the <module> in which it appears. A reference to an entity in which one specifies a global scope is valid only if the entity was defined as global and if the reference is from the same SQL-session in which it was defined. A reference to an entity in which one specifies a local scope is valid only if the entity was defined as local and if the reference is

from the same <module> in which it was defined. (The scope of non-extended statement names and non-extended cursor names is always local.)

Within an SQL-session, all global prepared statements (prepared statements with global statement names) belong to the SQL-session <module>. Within an SQL-session, each local prepared statement (prepared statements with local statement names) belongs to the <module> that contains the <prepare statement> or <execute immediate statement> with which it is prepared.

Note: The SQL-session <module> is defined in Subclause 4.39, "SQL-sessions".

Dynamic execution of SQL-statements can generally be accomplished in two different ways. Statements can be *prepared* for execution and then later executed one or more times; when the statement is no longer needed for execution, it can be *released* by the use of a <deallocate prepared statement>. Alternatively, a statement that is needed only once can be executed without the preparation step—it can be *executed immediately* (not all SQL-statements can be executed immediately).

Many SQL-statements can be written to use ''parameters'' (which are manifested in static execution of SQL-statements as parameters in <SQL procedure statement>s contained in <routine>s in <module>s or as host variables in <SQL procedure statement>s contained in <embedded SQL host program>s). In SQL-statements that are executed dynamically, the parameters are called dynamic parameters (<dynamic parameter specification>s) and are represented in SQL language by a <question mark> (?).

In many situations, an application that generates an SQL-statement for dynamic execution knows in detail the required characteristics (*e.g.*, <data type>, <length>, <precision>, <scale>, *etc.*) of each of the dynamic parameters used in the statement; similarly, the application may also know in detail the characteristics of the values that will be returned by execution of the statement. However, in other cases, the application may not know this information to the required level of detail; it is possible in some cases for the application to ascertain the information from the Information Schema, but in other cases (*e.g.*, when a returned value is derived from a computation instead of simply from a column in a table, or when dynamic parameters are supplied) this information is not generally available except in the context of preparing the statement for execution.

To provide the necessary information to applications, SQL permits an application to request the database system to *describe* a prepared statement. The description of a statement identifies the number of dynamic parameters (*describe input*) and their data type information or it identifies the number of values to be returned (*describe output*) and their data type information. The description of a statement is placed into the SQL descriptor areas already mentioned.

Many, but not all, SQL-statements can be prepared and executed dynamically.

Note: The complete list of statements that may be dynamically prepared and executed is defined in Subclause 4.31.4, "Preparable and immediately executable SQL-statements".

Certain ''set statements'' (<set catalog statement>, <set schema statement>, and <set names statement>) have no effect other than to set up default information (<catalog name>, <schema name>, and <character set>, respectively) to be applied to other SQL-statements that are prepared or executed immediately or that are invoked directly.

Syntax errors and Access Rule violations caused by the preparation or immediate execution of <preparable statement>s are identified when the statement is prepared (by <prepare statement>) or when it is executed (by <execute statement> or <execute immediate statement>); executed (by <execute statement> or <execute immediate statement>); such violations are indicated by the raising of an exception condition.

****Editor's Note****

The absence of Rules governing the relationship between prepared statements and SQL-sessions has been identified as a Possible Problem. See Possible Problem **167** in the Editor's Notes.

4.34 Direct invocation of SQL

Direct invocation of SQL is a mechanism for executing direct SQL-statements, known as <direct SQL statement>s. In direct invocation of SQL, the method of invoking <direct SQL statement>s, the method of raising conditions that result from the execution of <direct SQL statement>s, the method of accessing the diagnostics information that results from the execution of <direct SQL statement>s, and and the method of returning the results are implementation-defined.

4.35 Privileges and roles

A privilege authorizes a given category of <action> to be performed on a specified base table, view, column, domain, character set, collation, translation, abstract data type, type template, external function, or module or authorizes EXTERNAL PRIVILEGES on specified external functions by a specified <authorization identifier>. The mapping of <authorization identifier>s to operating system users is implementation-dependent. The <action>s that can be specified are:

— INSERT

— INSERT (<column name list>)

— UPDATE

— UPDATE (<column name list>)

— DELETE

— SELECT

— SELECT (<column name list>)

— REFERENCES

— REFERENCES (<column name list>)

— USAGE

— UNDER

— TRIGGER

— EXECUTE

A role may be granted to either <authorization identifier>s or to other roles with a <grant role statement>. The WITH ADMIN OPTION clause of the <grant role statement> specifies whether the recipient of a role may grant it to others. Each grant is represented and identified by a *role authorization descriptor*. A role authorization descriptor identifies the role, grantee, and whether the role was granted with the WITH ADMIN OPTION and hence is grantable.

An <authorization identifier> is specified for each schema and <module> as well as for each SQL-session.

A schema that is owned by a given <authorization identifier> may contain privilege descriptors that describe privileges granted to other <authorization identifier>s or to roles (grantees). The granted privileges apply to objects defined in the current schema. The WITH GRANT OPTION clause of a <grant statement> specifies whether the <authorization identifier> recipient of a privilege (acting as a grantor) may grant it to others.

When an SQL-session is initiated, the <authorization identifier> for the SQL-session, called the *SQL-session <authorization identifier>*, is determined in an implementation-dependent manner, unless the session is initiated using a <connect statement>. Subsequently, the SQL-session <authorization identifier> can be redefined by the successful execution of a <set session authorization identifier statement>.

A <module> may specify an <authorization identifier>, called a <module authorization identifier>. If the <module authorization identifier> is specified, then that <module authorization identifier> is used as the *current <authorization identifier>* for the execution of all <routine>s in the <module>. If the <module authorization identifier> is not specified, then the SQL-session <authorization identifier> is used as the current <authorization identifier> for the execution of each <routine> in the <module>.

A <schema definition> may specify an <authorization identifier>, called a <schema authorization identifier>. If the <schema authorization identifier> is specified, then that is used as the *current <authorization identifier>* for the creation of the schema. If the <schema authorization identifier> is not specified, then the <module authorization identifier> or the SQL-session <authorization identifier> is used as the current <authorization identifier> for the creation of the schema.

The current <authorization identifier> determines the privileges for the execution of each SQL-statement. For direct SQL, the SQL-session <authorization identifier> is always the current <authorization identifier>.

Each privilege is represented by a privilege descriptor. A privilege descriptor contains:

— the identification of the table, column, domain, character set, collation, translation, abstract data type, type template, external function, or module that the descriptor describes;

— the <authorization identifier> of the grantor of the privilege;

— the <authorization identifier> of the grantee of the privilege;

— identification of the action that the privilege allows; and

— an indication of whether or not the privilege is grantable.

A privilege descriptor with an action of INSERT, UPDATE, DELETE, SELECT, or REFERENCES is called a *table privilege descriptor* and identifies the existence of a privilege on the table identified by the privilege descriptor.

A privilege descriptor with an action of SELECT (<column name list>), INSERT (<column name list>), UPDATE (<column name list>), or REFERENCES (<column name list>) is called a *column privilege descriptor* and identifies the existence of a privilege on the columns in the table identified by the privilege descriptor.

A table privilege descriptor specifies that the privilege identified by the action (unless the action is DELETE) is to be automatically granted by the grantor to the grantee on all columns subsequently added to the table.

A privilege descriptor with an action of USAGE is called a usage privilege descriptor and identifies the existence of a privilege on the domain, abstract data type, type template, character set, collation, or translation identified by the privilege descriptor.

A privilege descriptor with an action of UNDER is called an under privilege descriptor and identifies the existence of the privilege on the abstract data type or table identified by the privilege descriptor. It authorizes the grantee to create one object UNDER another in a hierarchy, either of types or tables.

A privilege descriptor with an action of EXECUTE is called an execute privilege descriptor and identifies the existence of a privilege on the <module> identified by the privilege descriptor.

**** Editor's Note ****
Don't we need a paragraph describing TRIGGER privilege descriptors here?

A privilege descriptor that specifies EXTERNAL PRIVILEGES is called an external call privilege descriptor and identifies the existence of a privilege on the external function identified by the privilege descriptor.

Because roles may be granted to other roles, a role is said to ''contain'' other roles. The set of roles *X* contained in any role A is defined as the set of roles identified by role authorization descriptors as having been granted to *A*, together with all other roles contained by roles in *X*.

A grantable privilege is a privilege associated with a schema that may be granted by a <grant statement>.

The phrase *applicable roles* refers to the roles defined by role authorization descriptors as having been granted to the <authorization identifier> of the current SQL-session or to PUBLIC together with all other roles they contain.

Enabled applicable roles refers to the roles enabled by a <set role statement>. When an SQL-session is initiated, no roles are enabled. The execution of a <set role statement> disables any roles that may have been enabled by any previous <set role statement> and then enables the role specified in the statement, together with all roles contained in that role, for that user session.

The phrase *applicable privileges* refers to the privileges defined by the privilege descriptors that define privileges granted to the current <authorization identifier> or to the enabled applicable roles of that <authorization identifier>.

The set of applicable privileges for the current <authorization identifier> consists of the privileges defined by the privilege descriptors associated with that <authorization identifier> and the enabled applicable <privileges> of that <authorization identifier>, together with the privileges defined by the privilege descriptors associated with PUBLIC as well as privileges on the base tables, views, or domains contained in any schema owned by that <authorization identifier>.

Privilege descriptors that represent privileges for the owner of an object have a special grantor value, ''_SYSTEM''. This value is reflected in the Information Schema for all privileges that apply to the owner of the object.

4.36 SQL-agents

An SQL-agent is an implementation-dependent entity that causes the execution of SQL-statements.

4.37 SQL-transactions

An *SQL-transaction* (transaction) is a sequence of executions of SQL-statements that is atomic with respect to recovery. These operations are performed by one or more compilation units and <module>s or by the direct invocation of SQL.

It is implementation-defined whether or not the non-dynamic or dynamic execution of an SQL-data statement or the execution of an <SQL dynamic data statement> is permitted to occur within the same SQL-transaction as the non-dynamic or dynamic execution of an SQL-schema statement. If it does occur, then the effect on any open cursor, prepared dynamic statement, or deferred constraint is implementation-defined. There may be additional implementation-defined restrictions, requirements, and conditions. If any such restrictions, requirements, or conditions are violated, then an implementation-defined exception condition or a completion condition *warning* with an implementation-defined subclass code is raised.

Each <module> or direct invocation of SQL that executes an SQL-statement of an SQL-transaction is associated with that SQL-transaction. An SQL-transaction is initiated when no SQL-transaction is currently active and a <routine> is called that results in the execution of a *transaction-initiating* SQL-statement or by direct invocation of SQL that results in the execution of a transaction-initiating <direct SQL statement>. An SQL-transaction is terminated by a <commit statement> or a <rollback statement>. If an SQL-transaction is terminated by successful execution of a <commit statement>, then all changes made to SQL-data or schemas by that SQL-transaction are made persistent and accessible to all concurrent and subsequent SQL-transactions. If an SQL-transaction is terminated by a <rollback statement> or unsuccessful execution of a <commit statement>, then all changes made to SQL-data or schemas by that SQL-transaction are canceled. Committed changes cannot be canceled. If execution of a <commit statement> is attempted, but certain exception conditions are raised, it is unknown whether or not the changes made to SQL-data or schemas by that SQL-transaction are cancelled or made persistent.

An SQL-transaction may be partially rolled back by using a savepoint. The savepoint and its <savepoint name> are established within an SQL-transaction when a <savepoint statement> is executed. If a <rollback statement> references a savepoint, then all changes made to SQL-data or schema subsequent to the establishment of the savepoint are canceled, and the SQL-transaction is restored to its state as it was immediately following the execution of the <savepoint statement>. Savepoints are destroyed when an SQL-transaction is terminated, or when a <release savepoint statement> is executed. Savepoints may be redefined within an SQL-transaction by executing a <savepoint statement> which refers to a savepoint previously defined in the same SQL-transaction.

It is implementation-defined whether or not, or how, a <rollback statement> that references a <savepoint specifier> affects the contents of the diagnostics area, the contents of SQL descriptor areas, and the status of prepared statements.

An SQL-transaction has a *constraint mode* for each integrity constraint. The constraint mode for an integrity constraint in an SQL-transaction is described in Subclause 4.17, "Integrity constraints".

An SQL-transaction has an access mode that is either *read-only* or *read-write*. The access mode may be explicitly set by a <set transaction statement>; otherwise, it is implicitly set to *read-write*. The term *read-only* applies only to viewed tables and persistent base tables.

An SQL-transaction has a *diagnostics area limit*, which is a positive integer that specifies the maximum number of conditions that can be placed in the diagnostics area during execution of an SQL-statement in this SQL-transaction.

4.37 SQL-transactions

An SQL-transaction has a cursor mode that is either *cascade on* or *cascade off*. The cursor mode is initially *cascade on*. The cursor mode may be explicitly set by an option on the <open statement> that opens a cursor. Opening a cursor CASCADE OFF causes the checking of all table constraints and assertions affected by that cursor to be effectively deferred, the execution of all <referential action>s, pendant actions, and <triggered SQL statement>s affected by that cursor to be effectively deferred, and the fixing of all matching rows and unique matching rows. Closing the cursor causes an effective check of all table constraints and assertions, the effective execution of all referential actions, pendant actions, triggered SQL statements, and the re-evaluation of all matching rows and unique matching rows.

When the cursor mode of an SQL-transaction is *cascade off*, the SQL-transaction is said to be in *set-processing mode*. The interval between opening a cursor CASCADE OFF and closing that cursor is called the *set-processing mode session* for the SQL-transaction.

SQL-transactions initiated by different SQL-agents that access the same SQL-data or schemas and overlap in time are *concurrent SQL-transactions*.

An SQL-transaction has an *isolation level* that is READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, or SERIALIZABLE. The isolation level of an SQL-transaction defines the degree to which the operations on SQL-data or schemas in that SQL-transaction are affected by the effects of and can affect operations on SQL-data or schemas in concurrent SQL-transactions. The isolation level of a SQL-transaction is SERIALIZABLE by default. The level can be explicitly set by the <set transaction statement>.

The execution of concurrent SQL-transactions at isolation level SERIALIZABLE is guaranteed to be serializable. A serializable execution is defined to be an execution of the operations of concurrently executing SQL-transactions that produces the same effect as some serial execution of those same SQL-transactions. A serial execution is one in which each SQL-transaction executes to completion before the next SQL-transaction begins.

The isolation level specifies the kind of phenomena that can occur during the execution of concurrent SQL-transactions. The following phenomena are possible:

1) *P1* (''Dirty read''): SQL-transaction *T1* modifies a row. SQL-transaction *T2* then reads that row before *T1* performs a COMMIT. If *T1* then performs a ROLLBACK, *T2* will have read a row that was never committed and that may thus be considered to have never existed.

2) *P2* (''Non-repeatable read''): SQL-transaction *T1* reads a row. SQL-transaction *T2* then modifies or deletes that row and performs a COMMIT. If *T1* then attempts to reread the row, it may receive the modified value or discover that the row has been deleted.

3) *P3* (''Phantom''): SQL-transaction *T1* reads the set of rows *N* that satisfy some <search condition>. SQL-transaction *T2* then executes SQL-statements that generate one or more rows that satisfy the <search condition> used by SQL-transaction *T1*. If SQL-transaction *T1* then repeats the initial read with the same <search condition>, it obtains a different collection of rows.

The four isolation levels guarantee that each SQL-transaction will be executed completely or not at all, and that no updates will be lost. The isolation levels are different with respect to phenomena *P1*, *P2*, and *P3*. Table 9, "SQL-transaction isolation levels and the three phenomena" specifies the phenomena that are possible and not possible for a given isolation level.

Table 9—SQL-transaction isolation levels and the three phenomena

Level	*P1*	*P2*	*P3*
READ UNCOMMITTED	Possible	Possible	Possible
READ COMMITTED	Not Possible	Possible	Possible
REPEATABLE READ	Not Possible	Not Possible	Possible
SERIALIZABLE	Not Possible	Not Possible	Not Possible

Note: The exclusion of these phenomena for SQL-transactions executing at isolation level SERIALIZABLE is a consequence of the requirement that such transactions be serializable.

Changes made to SQL-data or schemas by an SQL-transaction in an SQL-session may be perceived by that SQL-transaction in that same SQL-session, and by other SQL-transactions, or by that same SQL-transaction in other SQL-sessions, at isolation level READ UNCOMMITTED, but cannot be perceived by other SQL-transactions at isolation level READ COMMITTED, REPEATABLE READ, or SERIALIZABLE until the former SQL-transaction terminates with a <commit statement>.

Regardless of the isolation level of the SQL-transaction, phenomena *P1*, *P2*, and *P3* shall not occur during the implied reading of schema definitions performed on behalf of executing an SQL-statement, the checking of integrity constraints, and the execution of referential actions associated with referential constraints. The schema definitions that are implicitly read are implementation-dependent. This does not affect the explicit reading of rows from tables in the Information Schema, which is done at the isolation level of the SQL-transaction.

The execution of a <rollback statement> may be initiated implicitly by an implementation when it detects the inability to guarantee the serializability of two or more concurrent SQL-transactions. When this error occurs, an exception condition is raised: *transaction rollback—serialization failure*.

The execution of a <rollback statement> may be initiated implicitly by an implementation when it detects unrecoverable errors. When such an error occurs, an exception condition is raised: *transaction rollback* with an implementation-defined subclass code.

The execution of an SQL-statement within an SQL-transaction has no effect on SQL-data or schemas other than the effect stated in the General Rules for that SQL-statement, in the General Rules for Subclause 11.8, "<referential constraint definition>", in the General Rules for Subclause 11.41, "<trigger definition>", and in the General Rules for Subclause 13.2, "<routine>". Together with serializable execution, this implies that all read operations are repeatable within an SQL-transaction at isolation level SERIALIZABLE, except for:

1) the effects of changes to SQL-data or schemas and its contents made explicitly by the SQL-transaction itself,

2) the effects of differences in parameter values supplied to procedures, and

3) the effects of references to time-varying system variables such as CURRENT_DATE and CURRENT_USER.

In some environments (e.g., remote database access), an SQL-transaction can be part of an encompassing transaction that is controlled by an agent other than the SQL-agent. The encompassing transaction may involve different resource managers, the SQL-environment being just one instance of such a manager. In such environments, an encompassing transaction must be terminated via that other agent, which in turn interacts with the SQL-environment via an interface that may be different from SQL (COMMIT or ROLLBACK), in order to coördinate the orderly termination of the encompassing transaction. When an SQL-transaction is part of an encompassing transaction that is controlled by an agent other than an SQL-agent and a <rollback statement> is initiated implicitly by an implementation, then the implementation will interact with that other agent to terminate that encompassing transaction. The specification of the interface between such agents and the SQL-environment is beyond the scope of this
ANSI American
ISO International
standard. However, it is important to note that the semantics of an SQL-transactaion remain as defined in the following sense:

— When an agent that is different from the SQL-agent requests the SQL-environment to roll-back an SQL-transaction, the General Rules of Subclause 16.7, "<rollback statement>", are performed.

— When such an agent requests the SQL-environment to commit an SQL-transaction, the General Rules of Subclause 16.6, "<commit statement>", are performed. To guarantee orderly termination of the encompassing transaction, this commit operation may be processed in several phases not visible to the application; not all the General Rules of Subclause 16.6, "<commit statement>", need to be executed in a single phase.

However, even in such environments, the SQL-agent interacts directly with the SQL-server to set attributes (such as *read-only* or *read-write*, isolation level, and constraints mode) that are specific to the SQL-transaction model.

4.38 SQL-connections

An *SQL-connection* is an association between an SQL-client and an SQL-server. An SQL-connection may be established and named by a <connect statement>, which identifies the desired SQL-server by means of an <SQL-server name>. A <connection name> is specified as a <simple value specification> whose value is an <identifier>. Two <connection name>s identify the same SQL-connection if their values, with leading and trailing <space>s removed, are equivalent according to the rules for <identifier> comparison in Subclause 5.2, "<token> and <separator>". It is implementation-defined how an implementation uses <SQL-server name> to determine the location, identity, and communication protocol required to access the SQL-server and create an SQL-session.

An SQL-connection is an *active SQL-connection* if any SQL-statement that initiates or requires an SQL-transaction has been executed at its SQL-server during the current SQL-transaction.

An SQL-connection is either *current* or *dormant*. If the SQL-connection established by the most recently executed implicit or explicit <connect statement> or <set connection statement> has not been terminated, then that SQL-connection is the *current SQL-connection*; otherwise, there is no current SQL-connection. An existing SQL-connection that is not the current SQL-connection is a *dormant SQL-connection*.

An SQL implementation may detect the loss of the current SQL-connection during execution of any SQL-statement. When such a connection failure is detected, an exception condition is raised: *connection exception—statement completion unknown*. This exception condition indicates that the results of the actions performed in the SQL-server on behalf of the statement are unknown to the SQL-agent.

Similarly, an SQL-implementation may detect the loss of the current SQL-connection during the execution of a <commit statement>. When such a connection failure is detected, an exception condition is raised: *connection exception—transaction resolution unknown*. This exception condition indicates that the SQL-implementation cannot verify whether the SQL-transaction was committed successfully, rolled back, or left active.

A user may initiate an SQL-connection between the SQL-client associated with the SQL-agent and a specific SQL-server by executing a <connect statement>. Otherwise, an SQL-connection between the SQL-client and an implementation-defined default SQL-server is initiated when a <routine> is called and no SQL-connection is current. The SQL-connection associated with an implementation-defined default SQL-server is called the *default SQL-connection*. An SQL-connection is terminated either by executing a <disconnect statement>, or following the last call to a <routine> within the last active <module>, or by the last execution of a <direct SQL statement> through the direct invocation of SQL. The mechanism and rules by which an SQL-environment determines whether a call to a <routine> is the last call within the last active <module> or the last execution of a <direct SQL statement> through the direct invocation of SQL are implementation-defined.

An implementation must support at least one SQL-connection and may require that the SQL-server be identified at the binding time chosen by the implementation. If an implementation permits more than one concurrent SQL-connection, then the SQL-agent may connect to more than one SQL-server and select the SQL-server by executing a <set connection statement>.

4.39 SQL-sessions

An *SQL-session* spans the execution of a sequence of consecutive SQL-statements invoked by a single user from a single SQL-agent or by the direct invocation of SQL.

An SQL-session is associated with an SQL-connection. The SQL-session associated with the default SQL-connection is called the *default SQL-session*. An SQL-session is either *current* or *dormant*. The *current SQL-session* is the SQL-session associated with the current SQL-connection. A *dormant SQL-session* is an SQL-session that is associated with a dormant SQL-connection.

An SQL-session has an SQL-session <module> that is different from any other <module> that exists simultaneously in the SQL-environment. The SQL-session <module> contains the global prepared SQL-statements that belong to the SQL-session. The SQL-session <module> contains a <module authorization clause> that specifies SCHEMA <schema name>, where the value of <schema name> is implementation-dependent.

Within an SQL-session,
ANSI module
ISO declared
local temporary tables are effectively created by <temporary table declaration>s and temporary views by <temporary view definition>s.
ANSI Module
ISO Declared
local temporary tables are accessible only to invocations of <routine>s in the <module> in which they are created. The definitions of
ANSI module

ISO declared
local temporary tables and temporary views persist until the end of the SQL-session.

An SQL-session has a unique implementation-dependent SQL-session identifier. This SQL-session identifier is different from the SQL-session identifier of any other concurrent SQL-session. The SQL-session identifier is used to effectively define implementation-defined schemas that contain the instances of any global temporary tables, created local temporary tables, or declared local temporary tables within the SQL-session.

An SQL-session has an <authorization identifier> that is initially set to an implementation-defined value when the SQL-session is started, unless the SQL-session is started as a result of successful execution of a <connect statement>, in which case the initial <authorization identifier> of the SQL-session is set to the value of the implicit or explicit <user name> contained in the <connect statement>.

An SQL-session has a default catalog name that is used to effectively qualify unqualified <schema name>s that are contained in <preparable statement>s when those statements are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement> or are contained in <direct SQL statement>s when those statements are invoked directly. The default catalog name is initially set to an implementation-defined value but can subsequently be changed by the successful execution of a <set catalog statement> or <set schema statement>.

An SQL-session has a default unqualified schema name that is used to effectively qualify unqualified <schema qualified name>s that are contained in <preparable statement>s when those statements are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement> or are contained in <direct SQL statement>s when those statements are invoked directly. The default unqualified schema name is initially set to an implementation-defined value but can subsequently be changed by the successful execution of a <set schema statement>.

An SQL-session has a default character set name that is used to identify the character set implicit for <identifier>s and <character string literal>s that are contained in <preparable statement>s when those statements are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement> or are contained in <direct SQL statement>s when those statements are invoked directly. The default character set name is initially set to an implementation-defined value but can subsequently be changed by the successful execution of a <set names statement>.

An SQL-session has a default local time zone displacement, which is a value of data type INTERVAL HOUR TO MINUTE. The default local time zone displacement is initially set to an implementation-defined value but can subsequently be changed by successful execution of a <set local time zone statement>.

An SQL-session has context that is preserved when an SQL-session is made dormant and restored when the SQL-session is made current. This context comprises:

— the current SQL-session identifier,

— the current <authorization identifier>,

— the identities of all instances of temporary tables,

— the SQL-session <module>,

— the current default catalog name,

— the current default unqualified schema name,

— the current character set name substitution value,

— the current default time zone,

— the current constraint mode for each integrity constraint,

— the current transaction access mode,

— the cursor position of all open cursors,

— the contents of all SQL dynamic descriptor areas,

— the current transaction isolation level, and

— the current transaction diagnostics area limit.

4.40 Client-server operation

Within an SQL-environment, an SQL-implementation may be considered to effectively contain an SQL-client component and one or more SQL-server components.

When an SQL-agent is active, it is bound in some implementation-defined manner to a single SQL-client. That SQL-client processes the explicit or implicit <SQL connection statement> for the first call to a <routine> by an SQL-agent. The SQL-client communicates with, either directly or possibly through other agents such as RDA, one or more SQL-servers. An SQL-session involves an SQL-agent, an SQL-client, and a single SQL-server.

<module>s associated with the SQL-agent exist in the SQL-environment containing the SQL-client associated with the SQL-agent.

Called <routine>s (and, analogously, <direct SQL statement>s) containing an <SQL connection statement> or an <SQL diagnostics statement> are processed by the SQL-client. Following the successful execution of a <connect statement> or a <set connection statement>, the <module>s associated with the SQL-agent are effectively materialized with an implementation-dependent <module name> in the SQL-server. Other called <routine>s and <direct SQL statement>s are processed by the SQL-server.

A call by the SQL-agent to a <routine> containing an <SQL diagnostics statement> fetches information from the diagnostics area associated with the SQL-client. Following the execution of an <SQL procedure statement> by an SQL-server, diagnostic information is passed in an implementation-dependent manner into the SQL-agent's diagnostics area in the SQL-client. The effect on diagnostic information of incompatibilities between the character repertoires supported by the SQL-client and SQL-server is implementation-dependent.

4.41 Information Schema

In each catalog in an SQL-environment, there is a schema, the Information Schema, with the name INFORMATION_SCHEMA, containing a number of view descriptors, one base table descriptor, and several domain descriptors. The data accessible through these views is a representation of all of the descriptors in all of the schemas in that catalog. The <query expression> of each view ensures that a given user can access only those rows of the view that represent descriptors on which he has privileges. The rows of each view are required to represent correctly the descriptors in the catalog as they existed at the start of the current SQL-transaction, as modified subsequently by any changes made by the current SQL-transaction. The SELECT privilege is granted on each of the Information Schema views to PUBLIC WITH GRANT OPTION so they can be queried by any user

and so that the SELECT privilege can be further granted on views that reference the Information Schema views. No further privilege is granted on them, so they cannot be updated.

The viewed tables in INFORMATION_SCHEMA are defined in terms of a collection of base tables in a schema named DEFINITION_SCHEMA, the Definition Schema. The only purpose of the definition of these base tables is to provide a data model to support the Information Schema. An implementation need do no more than simulate the existence of the base tables as viewed through the Information Schema views.

The Information Schema describes itself. It does not describe the base tables or views of the Definition Schema. If an implementation has defined additional objects that are associated with INFORMATION_SCHEMA, then those objects shall also be described in the Information Schema views.

4.42 Leveling

Four levels of conformance are specified in this
ANSI American
ISO International
Standard.

Entry SQL includes statements for defining schemas, data manipulation language, referential integrity, check constraints, and default clause from
ANSI ANSI X3.135-1992,
ISO ISO/IEC 9075:1992,
and options for module language and embedded SQL interfaces to seven different programming languages, as well as direct execution of the data manipulation statements. It also includes features related to deprecated features from
ANSI ANSI X3.135-1992
ISO ISO/IEC 9075:1992
(commas and parentheses in parameter lists, the SQLSTATE parameter, and renaming columns in the <select list>), features related to incompatibilities with
ANSI ANSI X3.135-1992
ISO ISO/IEC 9075:1992
(colons preceding <parameter name>s, WITH CHECK OPTION constraint clarifications), and aids for transitioning from
ANSI ANSI X3.135-1992
ISO ISO/IEC 9075:1992
to this
ANSI American
ISO International
Standard (<delimited identifier>s). Finally, it contains changes to correct defects found in
ANSI ANSI X3.135-1989
ISO ISO/IEC 9075:1989
(see Annex F, "Maintenance and interpretation of SQL").

Intermediate SQL includes major new facilities such as statements for changing schemas, dynamic SQL, isolation levels for SQL-transactions and named constraints. It also includes multiple-module support and cascade delete on referential actions, as well as numerous functional enhancements such as row and table expressions, union join, character string operations, table intersection and difference operations, simple domains, the CASE expression, casting between data types, a diagnostics management capability for data administration and more comprehensive error analysis, multiple character repertoires, interval and simplified datetime data types, and variable-length character strings. It also includes a requirement for a flagger facility to aid in writing portable

applications.

ISO Only–SQL3

Intermediate SQL provides CURRENT_USER as a feature to replace the deprecated feature USER. CURRENT_USER allows the returning of authorization identifiers based on alternative character repertoires.

Full SQL increases orthogonality and includes deferred constraint checking. Other technical enhancements include additional user options to define datetime data types, self-referencing updates and deletes, cascade update on referential actions, subqueries in check constraints, scrolled cursors, character translations, a bit string data type, temporary tables, additional referential constraint options, and simple assertions.

****Editor's Note****
This Subclause also must address the contents of the highest Level of SQL3, which should be given a proper name.

4.43 SQL Flagger

An SQL Flagger is an implementation-provided facility that is able to identify SQL language extensions, or other SQL processing alternatives, that may be provided by a conforming SQL-implementation (see Subclause 25.3, "Extensions and options"). An SQL Flagger is intended to assist SQL programmers in producing SQL language that is both portable and interoperable among different conforming SQL-implementations operating under different levels of this
ANSI American
ISO International
Standard.

An SQL Flagger is intended to effect a static check of SQL language. There is no requirement to detect extensions that cannot be determined until the General Rules are evaluated.

An SQL-implementation need only flag SQL language that is not otherwise in error as far as that implementation is concerned.

Note: If a system is processing SQL language that contains errors, then it may be very difficult within a single statement to determine what is an error and what is an extension. As one possibility, an implementation may choose to check SQL language in two steps; first through its normal syntax analyzer and secondly through the flagger. The first step produces error messages for nonstandard SQL language that the implementation cannot process or recognize. The second step processes SQL language that contains no errors as far as that implementation is concerned; it detects and flags at one time all nonstandard SQL language that could be processed by that implementation. Any such two-step process should be transparent to the end user.

In order to provide upward compatibility for its own customer base, or to provide performance advantages under special circumstances, a conforming SQL-implementation may provide user options to process conforming SQL language in a nonconforming manner. If this is the case, then it is required that the implementation also provide a flagger option, or some other implementation-defined means, to detect SQL conforming language that may be processed differently under the various user options. This flagger feature allows an application programmer to identify conforming SQL language that may perform differently in alternative processing environments provided by a conforming SQL-implementation. It also provides a valuable tool in identifying SQL elements that may have to be modified if SQL language is to be moved from a nonconforming to a conforming SQL processing environment.

X3H2-93-091 and YOK-003
4.43 SQL Flagger

An SQL Flagger provides one or more of the following ‘‘level of flagging’’ options:

— Entry SQL Flagging

— Intermediate SQL Flagging

— Full SQL Flagging

An SQL Flagger that provides one of these options shall be able to identify SQL language constructs that violate the indicated level of SQL language as defined in Subclause 4.42, "Leveling".

An SQL Flagger provides one or more of the following ‘‘extent of checking’’ options:

— Syntax Only

— Catalog Lookup

Under the Syntax Only option, the SQL Flagger analyzes only the SQL language that is presented; it checks for violations of any Syntax Rules that can be determined without access to the Information Schema.

Under the Catalog Lookup option, the SQL Flagger assumes the availability of Definition Schema information and checks for violations of all Syntax Rules and Access Rules, except Access Rules that deal with privileges. For example, some Syntax Rules place restrictions on data types; this flagger option would identify extensions that relax such restrictions. In order to avoid security breaches, this option may view the Definition Schema only through the eyes of a specific Information Schema.

5 Lexical elements

5.1 <SQL terminal character>

Function

Define the terminal symbols of the SQL language and the elements of strings.

Format

```
<SQL terminal character> ::=
          <SQL language character>
        | <SQL embedded language character>

<SQL embedded language character> ::=
          <left bracket>
        | <right bracket>

<SQL language character> ::=
          <simple Latin letter>
        | <digit>
        | <SQL special character>

<simple Latin letter> ::=
          <simple Latin upper case letter>
        | <simple Latin lower case letter>

<simple Latin upper case letter> ::=
              A | B | C | D | E | F | G | H | I | J | K | L | M | N | O
        | P | Q | R | S | T | U | V | W | X | Y | Z

<simple Latin lower case letter> ::=
              a | b | c | d | e | f | g | h | i | j | k | l | m | n | o
        | p | q | r | s | t | u | v | w | x | y | z

<digit> ::=
        0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<SQL special character> ::=
          <space>
        | <double quote>
        | <percent>
        | <ampersand>
        | <quote>
        | <left paren>
        | <right paren>
        | <asterisk>
        | <plus sign>
        | <comma>
        | <minus sign>
        | <period>
        | <solidus>
        | <colon>
        | <semicolon>
        | <less than operator>
```

```
        | <equals operator>
        | <greater than operator>
        | <question mark>
        | <left bracket>
        | <right bracket>
        | <circumflex>
        | <underscore>
        | <vertical bar>

<space> ::= !! space character in character set in use

<double quote> ::= "

<percent> ::= %

<ampersand> ::= &

<quote> ::= '

<left paren> ::= (

<right paren> ::= )

<asterisk> ::= *

<plus sign> ::= +

<comma> ::= ,

<minus sign> ::= -

<period> ::= .

<solidus> ::= /

<colon> ::= :

<semicolon> ::= ;

<less than operator> ::= <

<equals operator> ::= =

<greater than operator> ::= >

<question mark> ::= ?

<left bracket> ::= [

<right bracket> ::= ]

<circumflex> ::= ^

<underscore> ::= _

<vertical bar> ::= |
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) There is a one-to-one correspondence between the symbols contained in <simple Latin upper case letter> and the symbols contained in <simple Latin lower case letter> such that, for all *i*, the symbol defined as the *i*-th alternative for <simple Latin upper case letter> corresponds to the symbol defined as the *i*-th alternative for <simple Latin lower case letter>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

5.2 <token> and <separator>

Function

Specify lexical units (tokens and separators) that participate in SQL language.

Format

```
<token> ::=
          <nondelimiter token>
        | <delimiter token>

<nondelimiter token> ::=
          <regular identifier>
        | <key word>
        | <unsigned numeric literal>
        | <national character string literal>
        | <bit string literal>
        | <hex string literal>

<regular identifier> ::= <identifier body>

<identifier body> ::=
       <identifier start> [ { <underscore> | <identifier part> }... ]

<identifier start> ::= !! See the Syntax Rules

<identifier part> ::=
          <identifier start>
        | <digit>

<delimited identifier> ::=
       <double quote> <delimited identifier body> <double quote>

<delimited identifier body> ::= <delimited identifier part>...

<delimited identifier part> ::=
          <nondoublequote character>
        | <doublequote symbol>

<nondoublequote character> ::= !! See the Syntax Rules

<doublequote symbol> ::= <double quote><double quote>

<delimiter token> ::=
          <character string literal>
        | <date string>
        | <time string>
        | <timestamp string>
        | <interval string>
        | <delimited identifier>
        | <SQL special character>
        | <not equals operator>
        | <greater than or equals operator>
        | <less than or equals operator>
        | <concatenation operator>
        | <double period>
        | <double colon>
        | <assignment operator>
```

```
       | <left bracket>
       | <right bracket>

<not equals operator> ::= <>

<greater than or equals operator> ::= >=

<less than or equals operator> ::= <=

<concatenation operator> ::= ||

<double period> ::= ..

<double colon> ::= ::

<assignment operator> ::= :=

<separator> ::= { <comment> | <space> | <newline> }...

<comment> ::=
         <simple comment>
       | <bracketed comment>

<simple comment> ::=
       <simple comment introducer> [ <comment character>... ] <newline>

<simple comment introducer> ::= <minus sign><minus sign>[<minus sign>...]

<bracketed comment> ::=             !! (See the Syntax Rules)
       <bracketed comment introducer>
         <bracketed comment contents>
       <bracketed comment terminator>

<bracketed comment introducer> ::= <solidus><asterisk>

<bracketed comment terminator> ::= <asterisk><solidus>

<bracketed comment contents> ::=
       [ { <comment character> | <separator> }... ]

<comment character> ::=
         <nonquote character>
       | <quote>

<newline> ::= !! implementation-defined end-of-line indicator

<key word> ::=
         <reserved word>
       | <non-reserved word>

<non-reserved word> ::=
         ADA | ATOMIC

       | C | CATALOG_NAME | CHARACTER_SET_CATALOG | CHARACTER_SET_NAME
       | CHARACTER_SET_SCHEMA | CLASS_ORIGIN | COBOL | COLLATION_CATALOG | COLLATION_NAME

       | COLLATION_SCHEMA | COLUMN_NAME | COMMAND_FUNCTION
       | COMMITTED | CONDITION_NUMBER | CONNECTION_NAME | CONSTRAINT_CATALOG
       | CONSTRAINT_NAME | CONSTRAINT_SCHEMA | CURSOR_NAME
```

```
        | DATA | DATETIME_INTERVAL_CODE | DATETIME_INTERVAL_PRECISION | DYNAMIC_FUNCTION

        | FORTRAN
        | KEY_MEMBER | KEY_TYPE

        | LENGTH

        | MESSAGE_LENGTH | MESSAGE_OCTET_LENGTH | MESSAGE_TEXT | MORE | MUMPS

        | NAME | NULLABLE | NUMBER

        | PASCAL | PLI

        | REPEATABLE | RETURNED_LENGTH | RETURNED_OCTET_LENGTH | RETURNED_SQLSTATE
        ANSI Only--SQL3

        | ROUTINE_CATALOG | ROUTINE_NAME | ROUTINE_SCHEMA

        | ROW_COUNT

        | SCALE | SCHEMA_NAME | SERIALIZABLE | SERVER_NAME
        ANSI Only--SQL3

        | SPECIFIC_NAME

        | SUBCLASS_ORIGIN

        | TABLE_NAME | TYPE

        | UNCOMMITTED | UNNAMED

<reserved word> ::=
          ABSOLUTE | ACTION | ACTOR | ADD | AFTER | ALIAS
        | ALL | ALLOCATE | ALTER | AND | ANY | ARE
        | AS | ASC | ASSERTION | ASYNC | AT | AUTHORIZATION | AVG

        | BEFORE | BEGIN | BETWEEN | BIT | BIT_LENGTH | BOOLEAN | BOTH | BREADTH | BY

        | CALL | CASCADE | CASCADED | CASE | CAST | CATALOG | CHAR | CHARACTER
        | CHAR_LENGTH | CHARACTER_LENGTH | CHECK | CLASS | CLOSE | COALESCE | COLLATE
        | COLLATION | COLUMN | COMMIT | COMPLETION | CONNECT | CONNECTION | CONSTRAINT
        | CONSTRAINTS | CONSTRUCTOR | CONTINUE | CONVERT | CORRESPONDING | COUNT
        | CREATE | CROSS | CURRENT | CURRENT_DATE | CURRENT_TIME
        | CURRENT_TIMESTAMP | CURRENT_USER | CURSOR | CYCLE

        | DATA | DATE | DAY | DEALLOCATE | DEC | DECIMAL | DECLARE | DEFAULT
        | DEFERRABLE | DEFERRED | DELETE | DEPTH  | DEREF | DESC | DESCRIBE
        | DESCRIPTOR | DESTROY | DESTRUCTOR | DICTIONARY
        | DIAGNOSTICS | DISCONNECT | DISTINCT
        ANSI Only--SQL3
```

| DO

| DOMAIN | DOUBLE | DROP

| EACH

ANSI Only--SQL3

| ELEMENT

| ELSE | ELSEIF
| END | END-EXEC | EQUALS | ESCAPE | EXCEPT | EXCEPTION
| EXEC | EXECUTE | EXISTS | EXTERNAL | EXTRACT

| FALSE | FETCH | FIRST | FLOAT | FOR | FOREIGN | FOUND | FROM | FULL
| FUNCTION

| GENERAL | GET | GLOBAL | GO | GOTO | GRANT | GROUP

| HAVING | HOUR

| IDENTITY | IF | IGNORE | IMMEDIATE | IN | INDICATOR | INITIALLY | INNER | INOUT

| INPUT | INSENSITIVE | INSERT

ANSI Only--SQL3

| INSTEAD

| INT | INTEGER | INTERSECT | INTERVAL
| INTO | IS | ISOLATION

| JOIN

| KEY

| LANGUAGE | LAST | LEADING | LEAVE | LEFT
| LESS | LEVEL | LIKE | LIMIT

ANSI Only--SQL3

| LIST

| LOCAL | LOOP | LOWER

| MATCH | MAX | MIN | MINUTE | MODIFY | MODULE | MONTH

ANSI Only--SQL3

| MOVE | MULTISET

```
| NAMES | NATIONAL | NATURAL | NCHAR | NEW
```

ANSI Only--SQL3

```
| NEW_TABLE

| NEXT | NO
| NONE | NOT| NULL | NULLIF | NUMERIC

| OCTET_LENGTH | OF | OFF | OID
| OLD
```

ANSI Only--SQL3

```
| OLD_TABLE

| ON | ONLY | OPEN | OPERATION | OPERATORS
| OPTION | OR | ORDER | OTHERS | OUT | OUTER | OUTPUT | OVERLAPS

| PAD | PARAMETERS | PARTIAL | PENDANT | POSITION | PRECISION
| PREORDER | PREPARE | PRESERVE | PRIMARY | PRIOR | PRIVATE
| PRIVILEGES | PROCEDURE | PROTECTED | PUBLIC

| READ | REAL | RECURSIVE | REF
| REFERENCES | REFERENCING | RELATIVE | REPRESENTATION | RESIGNAL
| RESTRICT | RETURN | RETURNS | REVOKE | RIGHT | ROLE | ROLLBACK | ROUTINE
| ROW | ROWS

| SAVEPOINT | SCHEMA | SCROLL | SEARCH | SECOND | SECTION | SELECT | SENSITIVE
| SEQUENCE | SESSION | SESSION_USER | SET | SIGNAL
| SIMILAR | SIZE | SMALLINT | SOME | SPACE
```

ANSI Only--SQL3

```
| SPECIFIC

| SQL | SQLCODE
| SQLERROR | SQLEXCEPTION | SQLSTATE | SQLWARNING | STRUCTURE
| SUBSTRING | SUM | SYSTEM_USER

| TABLE | TEMPLATE | TEMPORARY | TEST | THAN
| THEN | THERE | TIME | TIMESTAMP | TIMEZONE_HOUR
| TIMEZONE_MINUTE | TO | TRAILING | TRANSACTION | TRANSLATE | TRANSLATION
| TRIGGER | TRIM | TRUE | TUPLE | TYPE

| UNDER | UNION | UNIQUE | UNKNOWN | UPDATE | UPPER | USAGE | USER | USING

| VALUE | VALUES | VARCHAR | VARIABLE | VARIANT | VARYING | VIEW
| VIRTUAL | VISIBLE

| WAIT | WHEN | WHENEVER | WHERE | WHILE | WITH | WITHOUT
| WORK | WRITE

| YEAR

| ZONE
```

Note: The list of <reserved word>s is considerably longer than the analogous list of <key word>s in
ANSI X3.135-1992.
ISO ISO 9075:1992.
To assist users of this
ANSI American
ISO International
Standard avoid such words in a possible future revision, the following list of potential <reserved word>s is provided. Readers must understand that there is no guarantee that all of these words will, in fact, become <reserved word>s in any future revision; furthermore, it is almost certain that additional words will be added to this list as any possible future revision emerges.

The words are: To Be Supplied

Syntax Rules

1) An <identifier start> is one of:

   a) A <simple Latin letter>; or

   b) A character that is identified as a letter in the character repertoire identified by the <module character set specification> or by the <character set specification>; or

   c) A character that is identified as a syllable in the character repertoire identified by the <module character set specification> or by the <character set specification>; or

   d) A character that is identified as an ideograph in the character repertoire identified by the <module character set specification> or by the <character set specification>.

2) With the exception of the <space> character explicitly contained in <timestamp string> and <interval string> and the permitted <separator>s in <bit string literal>s and <hex string literal>s, a <token>, other than a <character string literal>, a <national character string literal>, or a <delimited identifier>, shall not include a <space> character or other <separator>.

3) A <nondoublequote character> is one of:

   a) Any <SQL language character> other than a <double quote>;

   b) Any character other than a <double quote> in the character repertoire identified by the <module character set specification>; or

   c) Any character other than a <double quote> in the character repertoire identified by the <character set specification>.

4) The two <doublequote>s contained in a <doublequote symbol> shall not be separated by any <separator>.

5) Any <token> may be followed by a <separator>. A <nondelimiter token> shall be followed by a <delimiter token> or a <separator>.

   Note: If the Format does not allow a <nondelimiter token> to be followed by a <delimiter token>, then that <nondelimiter token> shall be followed by a <separator>.

6) There shall be no <space> nor <newline> separating the <minus sign>s of a <comment introducer>.

7) There shall be no <separator> between the <solidus> and <asterisk> of a <bracketed comment introducer> or between the <asterisk> and <solidus> of a <bracketed comment terminator>.

8) Within a <bracketed comment contents>, any <solidus> immediately followed by an <asterisk> without any intervening <separator> shall be considered to be the <bracketed comment introducer> of a <separator> that is a <bracketed comment>.

Note: Conforming programs should not place <simple comment> within a <bracketed comment> because if such a <simple comment> contains the sequence of characters ''*/'' without a preceding ''/*'' in the same <simple comment>, it will prematurely terminate the containing <bracketed comment>.

9) SQL text containing one or more instances of <comment> is equivalent to the same SQL text with the <comment> replaced with <newline>.

10) The sum of the number of <identifier start>s and the number of <identifier part>s in a <regular identifier> shall not be greater than 128.

11) The <delimited identifier body> of a <delimited identifier> shall not comprise more than 128 <delimited identifier part>s.

12) The <identifier body> of a <regular identifier> is equivalent to an <identifier body> in which every letter that is a lower-case letter is replaced by the equivalent upper-case letter or letters. This treatment includes determination of equivalence, representation in the Information and Definition Schemas, representation in the diagnostics area, and similar uses.

13) The <identifier body> of a <regular identifier> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter or letters), treated as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT, shall not be equal, according to the comparison rules in Subclause 8.2, "<comparison predicate>", to any <reserved word> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter or letters), treated as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT.

Note: It is the intention that no <key word> specified in this
ANSI American
ISO International
Standard or revisions thereto shall end with an <underscore>.

14) Two <regular identifier>s are equivalent if their <identifier body>s, considered as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT, compare equally according to the comparison rules in Subclause 8.2, "<comparison predicate>".

15) A <regular identifier> and a <delimited identifier> are equivalent if the <identifier body> of the <regular identifier> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter or letters) and the <delimited identifier body> of the <delimited identifier> (with all occurrences of <quote> replaced by <quote symbol> and all occurrences of <double-quote symbol> replaced by <double quote>), considered as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT and an implementation-defined collation that is sensitive to case, compare equally according to the comparison rules in Subclause 8.2, "<comparison predicate>".

16) Two <delimited identifier>s are equivalent if their <delimited identifier body>s (with all occurrences of <quote> replaced by <quote symbol> and all occurrences of <doublequote symbol> replaced by <doublequote>), considered as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT and an implementation-defined collation that is sensitive to case, compare equally according to the comparison rules in Subclause 8.2, "<comparison predicate>".

17) For the purposes of identifying <key word>s, any <simple Latin lower case letter> contained in a candidate <key word> shall be effectively treated as the corresponding <simple Latin upper case letter>.

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) No <identifier body> shall end in an <underscore>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) No <regular identifier> or <delimited identifier body> shall contain more than 18 <character representation>s.

   b) An <identifier body> shall contain no <simple Latin lower case letter>.

5.3 <literal>

Function

Specify a non-null value.

Format

```
<literal> ::=
        <signed numeric literal>
      | <general literal>

<unsigned literal> ::=
        <unsigned numeric literal>
      | <general literal>

<general literal> ::=
        <character string literal>
      | <national character string literal>
      | <bit string literal>
      | <hex string literal>
      | <datetime literal>
      | <interval literal>
      | <enumeration literal>
      | <boolean literal>
```

ANSI Only--SQL3

```
      | <oid literal>
      | <collection literal>
```

```
<character string literal> ::=
      [ <introducer><character set specification> ]
      <quote> [ <character representation>... ] <quote>
        [ { <separator>... <quote> [ <character representation>... ] <quote> }... ]

<introducer> ::= <underscore>

<character representation> ::=
        <nonquote character>
      | <quote symbol>

<nonquote character> ::= !! See the Syntax Rules.

<quote symbol> ::= <quote><quote>

<national character string literal> ::=
      N <quote> [ <character representation>... ] <quote>
        [ { <separator>... <quote> [ <character representation>... ] <quote> }... ]

<bit string literal> ::=
      B <quote> [ <bit>... ] <quote>
        [ { <separator>... <quote> [ <bit>... ] <quote> }... ]

<hex string literal> ::=
      X <quote> [ <hexit>... ] <quote>
```

```
        [ { <separator>... <quote> [ <hexit>... ] <quote> }... ]

<bit> ::= 0 | 1

<hexit> ::= <digit> | A | B | C | D | E | F | a | b | c | d | e | f

<signed numeric literal> ::=
       [ <sign> ] <unsigned numeric literal>

<unsigned numeric literal> ::=
         <exact numeric literal>
       | <approximate numeric literal>

<exact numeric literal> ::=
         <unsigned integer> [ <period> [ <unsigned integer> ] ]
       | <period> <unsigned integer>

<sign> ::= <plus sign> | <minus sign>

<approximate numeric literal> ::= <mantissa> E <exponent>

<mantissa> ::= <exact numeric literal>

<exponent> ::= <signed integer>

<signed integer> ::= [ <sign> ] <unsigned integer>

<unsigned integer> ::= <digit>...

<enumeration literal> ::=
       <domain name> <double colon> <enumeration name>

<datetime literal> ::=
         <date literal>
       | <time literal>
       | <timestamp literal>

<date literal> ::=
       DATE <date string>

<time literal> ::=
       TIME <time string>

<timestamp literal> ::=
       TIMESTAMP <timestamp string>

<date string> ::=
       <quote> <date value> <quote>

<time string> ::=
       <quote> <time value> [ <time zone interval> ] <quote>

<timestamp string> ::=
       <quote> <date value> <space> <time value> [ <time zone interval> ] <quote>

<time zone interval> ::=
       <sign> <hours value> <colon> <minutes value>

<date value> ::=
```

```
       <years value> <minus sign> <months value> <minus sign> <days value>

<time value> ::=
       <hours value> <colon> <minutes value> <colon> <seconds value>

<interval literal> ::=

       INTERVAL [ <sign> ] <interval string> <interval qualifier>

<interval string> ::=
       <quote> { <year-month literal> | <day-time literal> } <quote>

<year-month literal> ::=
         <years value>
       | [ <years value> <minus sign> ] <months value>

<day-time literal> ::=
         <day-time interval>
       | <time interval>

<day-time interval> ::=
       <days value>
         [ <space> <hours value> [ <colon> <minutes value> [ <colon> <seconds value> ] ] ]


<time interval> ::=
         <hours value> [ <colon> <minutes value> [ <colon> <seconds value> ] ]
       | <minutes value> [ <colon> <seconds value> ]
       | <seconds value>

<years value> ::= <datetime value>

<months value> ::= <datetime value>

<days value> ::= <datetime value>

<hours value> ::= <datetime value>

<minutes value> ::= <datetime value>

<seconds value> ::=
         <seconds integer value> [ <period> [ <seconds fraction> ] ]

<seconds integer value> ::= <unsigned integer>

<seconds fraction> ::= <unsigned integer>

<datetime value> ::= <unsigned integer>

<boolean literal> ::=
         TRUE
       | FALSE
```

ANSI Only--SQL3

```
<oid literal> ::= OID <oid string>

<oid string> ::= <quote> <oid value> <quote>
```

```
<oid value> ::= <character representation>...

<collection literal> ::=
        <set value constructor>
      | <multiset value constructor>
      | <list value constructor>
```

Syntax Rules

1) In a <character string literal> or <national character string literal>, the sequence:

   <quote> <character representation>... <quote>
   <separator>... <quote> <character representation>... <quote>

   is equivalent to the sequence

   <quote> <character representation>... <character representation>... <quote>

   Note: The <character representation>s in the equivalent sequence are in the same sequence and relative sequence as in the original <character string literal>.

2) In a <bit string literal>, the sequence

   <quote> <bit>... <quote> <separator>... <quote> <bit>... <quote>

   is equivalent to the sequence

   <quote> <bit>... <bit>... <quote>

   Note: The <bit>s in the equivalent sequence are in the same sequence and relative sequence as in the original <bit string literal>.

3) In a <hex string literal>, the sequence

   <quote> <hexit>... <quote> <separator>... <quote> <hexit>... <quote>

   is equivalent to the sequence

   <quote> <hexit>... <hexit>... <quote>

   Note: The <hexit>s in the equivalent sequence are in the same sequence and relative sequence as in the original <hex string literal>.

4) In a <character string literal>, <national character string literal>, <bit string literal>, or <hex string literal>, a <separator> shall contain a <newline>.

5) A <nonquote character> is one of:

   a) Any <SQL language character> other than a <quote>;

   b) Any character other than a <quote> in the character repertoire identified by the <module character set specification>; or

   c) Any character other than a <quote> in the character repertoire identified by the <character set specification> or implied by ''N''.

6) If a <character set specification> is not specified in a <character string literal>, then the set of characters contained in the <character string literal> shall be wholly contained in either <SQL language character> or the character repertoire indicated by:

Case:

a) If the <character string literal> is contained in a <module>, then the <module character set specification>,

b) If the <character string literal> is contained in a <schema definition> that is not contained in a <module>, then the <schema character set specification>,

c) If the <character string literal> is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default character set name for the SQL-session.

7) If a <character set specification> is specified in a <character string literal>, then

a) There shall be no <separator> between the <introducer> and the <character set specification>.

b) The set of characters contained in the <character string literal> shall be wholly contained in the character repertoire indicated by the <character set specification>.

8) A <national character string literal> is equivalent to a <character string literal> with the ''N'' replaced by ''<introducer><character set specification>'', where ''<character set specification>'' is an implementation-defined <character set name>.

9) The data type of a <character string literal> is fixed-length character string. The length of a <character string literal> is the number of <character representation>s that it contains. Each <quote symbol> contained in <character string literal> represents a single <quote> in both the value and the length of the <character string literal>. The two <quote>s contained in a <quote symbol> shall not be separated by any <separator>.

Note: <character string literal>s are allowed to be zero-length strings (*i.e.*, to contain no characters) even though it is not permitted to declare a <data type> that is CHARACTER with <length> zero.

10) The data type of a <bit string literal> is fixed-length bit string. The length of a <bit string literal> is the number of bits that it contains.

11) The data type of a <hex string literal> is fixed-length bit string. Each <hexit> appearing in the literal is equivalent to a quartet of bits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F are interpreted as 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, and 1111, respectively. The <hexit>s a, b, c, d, e, and f have respectively the same values as the <hexit>s A, B, C, D, E, and F.

12) An <exact numeric literal> without a <period> has an implied <period> following the last <digit>.

13) The data type of an <exact numeric literal> is exact numeric. The precision of an <exact numeric literal> is the number of <digit>s that it contains. The scale of an <exact numeric literal> is the number of <digit>s to the right of the <period>.

14) The data type of an <approximate numeric literal> is approximate numeric. The precision of an <approximate numeric literal> is the precision of its <mantissa>.

15) The <domain name> of an <enumeration literal> shall identify a <domain definition> whose <data type> is an <enumerated type> that includes the <enumeration name> of the <enumeration literal>. That <domain> and <data type> are the <domain> and <data type> of the <enumeration literal>.

16) The data type of a <date literal> is DATE.

17) The data type of a <time literal> that does not specify <time zone interval> is TIME(*P*), where *P* is the number of digits in <seconds fraction>, if specified, and 0 otherwise. The data type of a <time literal> that specifies <time zone interval> is TIME(*P*) WITH TIME ZONE, where *P* is the number of digits in <seconds fraction>, if specified, and 0 otherwise.

18) The data type of a <timestamp literal> that does not specify <time zone interval> is TIMESTAMP(*P*), where *P* is the number of digits in <seconds fraction>, if specified, and 0 otherwise. The data type of a <timestamp literal> that specifies <time zone interval> is TIMESTAMP(*P*) WITH TIME ZONE, where *P* is the number of digits in <seconds fraction>, if specified, and 0 otherwise.

19) If <time zone interval> is not specified, then the effective <time zone interval> of the datetime data type is the current default time zone displacement for the SQL-session.

20) Let *datetime component* be either <years value>, <months value>, <days value>, <hours value>, <minutes value>, or <seconds value>.

21) Let *N* be the number of <datetime field>s in the precision of the <interval literal>, as specified by <interval qualifier>.

The <interval literal> being defined shall contain *N* datetime components.

The data type of <interval literal> specified with an <interval qualifier> is INTERVAL with the <interval qualifier>.

22) Within a <datetime literal>, the <years value> shall contain four digits. The <seconds integer value> and other datetime components, with the exception of <seconds fraction>, shall each contain two digits.

23) Within the definition of a <datetime literal>, the <datetime value>s are constrained by the natural rules for dates and times according to the Gregorian calendar.

24) Within the definition of an <interval literal>, the <datetime value>s are constrained by the natural rules for intervals according to the Gregorian calendar.

25) Within the definition of a <year-month literal>, the <interval qualifier> shall not specify DAY, HOUR, MINUTE, or SECOND. Within the definition of a <day-time literal>, the <interval qualifier> shall not specify YEAR or MONTH.

26) Within the definition of a <datetime literal>, the value of the <time zone interval> shall be in the range −12:59 to +13:00.

27) The null class of a <literal> is the general null class.

ANSI Only–SQL3

28) The <oid value> of an <oid literal> represents an object identifier. Each <quote symbol> contained in an <oid value> represents a single <quote> in the value. The two <quote>s contained in a <quote symbol> shall not be separated by any <separator>. The method of representing an object identifier as an <oid value> is implementation-dependent.

29) The data type of an <oid literal> is the data type of the abstract data type that its <oid value> represents.

30) Within the definition of an <oid literal>, the allowed <oid value>s are implementation-dependent.

31) The data type of a <collection literal> is the collection type of the <set value constructor>, <multiset value constructor>, or <list value constructor> it immediately contains.

Access Rules

None.

General Rules

1) The value of a <character string literal> is the sequence of <character representation>s that it contains.

2) The value of a <bit string literal> or a <hex string literal> is the sequence of bits that it contains.

3) The numeric value of an <exact numeric literal> is determined by the normal mathematical interpretation of positional decimal notation.

4) The numeric value of an <approximate numeric literal> is approximately the product of the exact numeric value represented by the <mantissa> with the number obtained by raising the number 10 to the power of the exact numeric value represented by the <exponent>.

5) The <sign> in a <signed numeric literal> or an <interval literal> is a monadic arithmetic operator. The monadic arithmetic operators + and − specify monadic plus and monadic minus, respectively. If neither monadic plus nor monadic minus are specified in a <signed numeric literal> or an <interval literal>, or if monadic plus is specified, then the literal is positive. If monadic minus is specified in a <signed numeric literal> or <interval literal>, then the literal is negative.

6) The value of an <enumeration literal> is its enumeration value.

7) Let *V* be the integer value of the <unsigned integer> contained in <seconds fraction> and let *N* be the number of digits in the <seconds fraction> respectively. The resultant value of the <seconds fraction> is effectively determined as follows:

   Case:

   a) If <seconds fraction> is specified within the definition of a <datetime literal>, then the effective value of the <seconds fraction> is $V * 10^{-N}$ seconds.

   b) If <seconds fraction> is specified within the definition of an <interval literal>, then let *M* be the <interval fractional seconds precision> specified in the <interval qualifier>.

      Case:

      i) If $N < M$, then let *V1* be $V * 10^{M-N}$; the effective value of the <seconds fraction> is $V1 * 10^{-M}$ seconds.

      ii) If $N > M$, then let *V2* be the integer part of the quotient of $V / 10^{N-M}$; the effective value of the <seconds fraction> is $V 2 * 10^{-M}$ seconds.

iii) Otherwise, the effective value of the <seconds fraction> is $V * 10^{-M}$ seconds.

8) The *i*-th datetime component in a <datetime literal> or <interval literal> assigns the value of the datetime component to the *i*-th <datetime field> in the <datetime literal> or <interval literal>.

9) If <time zone interval> is specified, then the time and timestamp values in <time literal> and <timestamp literal> represent a datetime in the specified time zone. Otherwise, the time and timestamp values represent a datetime in the current default time zone of the SQL-session. The value of the <time literal> or the <timestamp literal> is effectively the <time value> or the <date value> and <time value> together minus the <time zone interval> value, followed by the <time zone interval>.

Note: <time literal>s and <timestamp literal>s are specified in a time zone chosen by the SQL-agent (the default is the current default time zone of the SQL-session). However, they are effectively converted to UTC while maintaining the <time zone interval> information that permits knowing the original time zone value for the time or timestamp value.

ANSI Only–SQL3

10) The value of an <oid literal> is that instance of the abstract data type identified by the <oid value> of that <oid literal>.

11) The value of a <collection literal> is the value of the <set value constructor>, <multiset value constructor>, or <list value constructor> it immediately contains.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <general literal> shall not be an <enumeration literal>.

b) A <general literal> shall not be a <boolean literal>.

ANSI Only–SQL3

c) A <general literal> shall not be an <oid literal>.

d) A <general literal> shall not specify a <collection literal>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) An <unsigned integer> that is a <seconds fraction> shall not contain more than 6 <digit>s.

b) A <general literal> shall not be a <bit string literal> or a <hex string literal>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <general literal> shall not be a <national character string literal>.

   b) A <general literal> shall not be a <datetime literal> or <interval literal>.

   c) A <character string literal> shall contain at least one <character representation>.

   d) Conforming Entry SQL language shall contain exactly one repetition of <character representation> (that is, it shall contain exactly one sequence of ''<quote> <character representation>... <quote>'').

   e) A <character string literal> shall not specify a <character set specification>.

5.4 Names and identifiers

Function

Specify names.

Format

```
<identifier> ::=
       [ <introducer><character set specification> ] <actual identifier>

<actual identifier> ::=
         <regular identifier>
       | <delimited identifier>

<SQL language identifier> ::=
       <SQL language identifier start>
         [ { <underscore> | <SQL language identifier part> }... ]

<SQL language identifier start> ::= <simple Latin letter>

<SQL language identifier part> ::=
         <simple Latin letter>
       | <digit>

<authorization identifier> ::= <identifier>

<table name> ::=
```

ANSI Only--SQL3

```
         <local or schema qualified name>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
         <schema qualified name>
       | <qualified local table name>

<qualified local table name> ::=
       MODULE <period> <local table name>

<local table name> ::= <qualified identifier>
```

```
<domain name> ::= <schema qualified name>

<enumeration name> ::= <identifier>

<schema name> ::=
       [ <catalog name> <period> ] <unqualified schema name>

<unqualified schema name> ::= <identifier>

<catalog name> ::= <identifier>

<schema qualified name> ::=
```

```
       [ <schema name> <period> ] <qualified identifier>
```

ANSI Only--SQL3

```
<local or schema qualified name> ::=
       [ <local or schema qualifier> <period> ] <qualified identifier>

<local or schema qualifier> ::=
         <schema name>
       | MODULE
       | <statement label>
```

```
<qualified identifier> ::= <identifier>

<column name> ::=
         <identifier>
       | OID

<correlation name> ::= <identifier>

<module name> ::=
         <schema qualified name>
       | <identifier>
```

ANSI Only--SQL3

```
<cursor name> ::= <local qualified name>

<local qualified name> ::=
       [ <local qualifier> <period> ] <qualified identifier>

<local qualifier> ::=
         <module name>
       | <statement label>

<routine name> ::= <local or schema qualified name>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
<cursor name> ::= <identifier>

<routine name> ::=
         <identifier>
       | <schema qualified name>
```

ANSI Only--SQL3

```
       <specific name> ::= <schema qualified name>
```

```
<SQL statement name> ::=
         <statement name>
```

```
       | <extended statement name>

<statement name> ::= <identifier>

<extended statement name> ::=
       [ <scope option> ] <simple value specification>

<dynamic cursor name> ::=
         <cursor name>
       | <extended cursor name>

<extended cursor name> ::=
       [ <scope option> ] <simple value specification>

<descriptor name> ::=
       [ <scope option> ] <simple value specification>

<scope option> ::=
         GLOBAL
       | LOCAL

<parameter name> ::= <colon> <identifier>

<type template name> ::= <schema qualified name>

<template parameter name> ::= <colon> <identifier>

<SQL variable name> ::= <colon> <identifier>

<constraint name> ::= <schema qualified name>

<external routine name> ::= <identifier>

<trigger name> ::= <schema qualified name>

<collation name> ::= <schema qualified name>

<character set name> ::= [ <schema name> <period> ] <SQL language identifier>

<translation name> ::= <schema qualified name>

<form-of-use conversion name> ::= <schema qualified name>

<abstract data type name> ::= <schema qualified name>

<component name> ::= <identifier>

<savepoint name> ::= <identifier>

<role name> ::= <identifier>

<connection name> ::= <simple value specification>

<null class name> ::= <schema qualified name>

<exception name> ::= <identifier>

<SQL-server name> ::= <simple value specification>
```

```
<user name> ::= <simple value specification>
```

Syntax Rules

1) If a <character set specification> is not specified in an <identifier>, then the set of characters contained in the <identifier> shall be wholly contained in either <SQL language character> or the character repertoire indicated by:

   Case:

   a) If the <identifier> is contained in a <module>, then the <module character set specification>,

   b) If the <identifier> is contained in a <schema definition> that is not contained in a <module>, then the <schema character set specification>,

   c) If the <identifier> is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default character set name for the SQL-session.

2) If a <character set specification> is specified in an <identifier>, then:

   a) There shall be no <separator> between the <introducer> and the <character set specification>.

   b) The set of characters contained in the <identifier body> or <delimited identifier body> shall be wholly contained in the character repertoire indicated by the <character set specification>.

3) The sum of the number of <SQL language identifier start>s and the number of <SQL language identifier part>s in an <SQL language identifier> shall not be greater than 128.

4) An <SQL language identifier> is equivalent to an <SQL language identifier> in which every letter that is a lower-case letter is replaced by the equivalent upper-case letter or letters. This treatment includes determination of equivalence, representation in the Information and Definition Schemas, representation in the diagnostics area, and similar uses.

5) An <SQL language identifier> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter), treated as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT, shall not be equal, according to the comparison rules in Subclause 8.2, "<comparison predicate>", to any <reserved word> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter), treated as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT.

   Note: It is the intention that no <key word> specified in this
   ANSI American
   ISO International
   standard or revisions thereto shall end with an <underscore>.

ANSI Only–SQL3

---

6) If a <table name> *TN* with a <qualified identifier> *QI* does not contain a <local or schema qualifier>, then

Case:

a) If *TN* is simply contained in a <compound statement> *CS* whose <local declaration list> contains a <temporary table declaration> or <temporary view declaration> *TT* whose declared local name is equal to *QI*, then the <statement label> of *CS* is the implicit <local or schema qualifier> of *TN*.

b) If *TN* is contained in a <module> *M* whose <module contents> contain a <temporary table declaration> or <temporary view declaration> *TT* whose <table name> *TN* has a <qualified identifier> equal to *QI*, then the <module name> of *M* is the implicit <local or schema qualifier> of *TN*.

c) Otherwise, *TN* shall be contained in a <schema definition> *S* that contains a <table definition> or <view definition> whose <table name> has a <qualified identifier> equal to *QI*. The <schema name> of *S* is the implicit <local or schema qualifier> of *TN*.

7) If a <table name> *TN* with a <qualified identifier> *QI* contains a <local or schema qualifier> *LSQ*, then

Case:

a) If *TN* is contained in a <compound statement> *CS* whose <statement label> is equal to *LSQ*, then the <local declaration list> of *CS* shall contain a <temporary table declaration> or <temporary view declaration> *TT* whose declared local name is equal to *QI*.

b) If *TN* is contained in a <module> *M* whose <module name> is *LSQ*, or if *LSQ* is ''MODULE'', then the <module contents> of *M* shall contain a <temporary table declaration> or <temporary view declaration> *TT* whose <table name> has a <qualified identifier> equal to *QI*.

c) Otherwise, *LSQ* shall be a <schema name> that designates a <schema> that contains a <table definition> or <view definition> whose <table name> has a <qualified identifier> equal to *QI*.

8) If a <cursor name> *CN* with a <qualified identifier> *QI* does not contain a <local qualifier>, then

Case:

a) If *CN* is simply contained in a <compound statement> *CS* whose <local declaration list> contains a <declare cursor> whose declared local name is equal to *QI*, then the <statement label> of *CS* is the implicit <local qualifier> of *CN*.

b) Otherwise, *CN* shall be contained in a <module> *M* whose <module contents> contain a <declare cursor> whose <cursor name> *CN* has a <qualified identifier> equal to *QI* and the <module name> of *M* is the implicit <local qualifier> of *CN*.

9) If a <cursor name> *CN* with a <qualified identifier> *QI* contains a <local qualifier> *LQ*, then

Case:

a) If *CN* is contained in a <compound statement> *CS* whose <statement label> is equal to *LQ*, then the <local declaration list> of *CD* shall contain a <declare cursor> whose declared local name is equal to *QI*.

b) Otherwise, *CN* shall be contained in a <module> *M* whose <module name> is *LQ* and whose <module contents> contain a <declare cursor> whose <cursor name> is *CN*.

ISO Only—caused by ANSI changes not yet considered by ISO

10) If <table name> is not a <qualified local table name>, then the table identified by <table name> shall not be a declared local temporary table.

11) No <unqualified schema name> shall specify DEFINITION_SCHEMA.

12) If a <schema qualified name> does not contain a <schema name>, then

Case:

a) If the <schema qualified name> is contained in a <schema definition>, then the <schema name> that is specified or implicit in the <schema definition> is implicit.

b) If the <schema qualified name> is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default <unqualified schema name> for the SQL-session is implicit.

c) Otherwise, the <schema name> that is specified or implicit for the <module> is implicit.

13) If a <schema name> does not contain a <catalog name>, then

Case:

a) If the <unqualified schema name> is contained in a <module authorization clause>, then an implementation-defined <catalog name> is implicit.

b) If the <unqualified schema name> is contained in a <schema definition> other than in a <schema name clause>, then the <catalog name> that is specified or implicit in the <schema name clause> is implicit.

c) If the <unqualified schema name> is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default catalog name for the SQL-session is implicit.

d) If the <unqualified schema name> is contained in a <schema name clause>, then

Case:

i) If the <schema name clause> is contained in a <module>, then the explicit or implicit <catalog name> contained in the <module authorization clause> is implicit.

ii) Otherwise, an implementation-defined <catalog name> is implicit.

e) Otherwise, the explicit or implicit <catalog name> contained in the <module authorization clause> is implicit.

14) Two <schema qualified name>s are equal if and only if they have the same <qualified identifier> and the same <schema name>, regardless of whether the <schema name>s are implicit or explicit.

15) Two <schema name>s are equal if and only if they have the same <unqualified schema name> and the same <catalog name>, regardless of whether the <catalog name>s are implicit or explicit.

16) An <identifier> that is a <correlation name> is associated with a table within a particular scope. The scope of a <correlation name> is either a <select statement: single row>, <subquery>, or <query specification> (see Subclause 6.5, "<table reference>"), or is a <query term> that contains a <recursive union> (see Subclause 7.14, "<recursive union>"), or is a <trigger definition> (see Subclause 11.41, "<trigger definition>"). Scopes may be nested. In different scopes, the same <correlation name> may be associated with different tables or with the same table.

17) The <simple value specification> of <extended statement name> or <extended cursor name> shall not be a <literal>.

18) The data type of the <simple value specification> of <extended statement name> shall be character string with an implementation-defined character set and shall have an octet length of 128 octets or less.

19) The data type of the <simple value specification> of <extended cursor name> shall be character string with an implementation-defined character set and shall have an octet length of 128 octets or less.

20) If a <module name> is contained in a <module> that is created by a <create module statement> or if it identifies such a <module>, then the <module name> shall be a <schema qualified name>; otherwise, the <module name> shall be an <identifier>.

21) The data type of the <simple value specification> of <descriptor name> shall be character string with an implementation-defined character set and shall have an octet length of 128 octets or less.

22) In a <descriptor name>, <extended statement name>, or <extended cursor name>, if a <scope option> is not specified, then a <scope option> of LOCAL is implicit.

23) No <authorization identifier> shall specify ''PUBLIC''.

24) Those <identifier>s that are valid <authorization identifier>s are implementation-defined.

25) A <template parameter name> shall appear in a <type template definition>.

26) Any <template parameter name> contained in the <abstract data type body> of a <type template definition> shall also be contained in a <template parameter declaration> of the same <type template definition>.

Case:

a) A <template parameter name> that is contained in <data type> shall be contained in a <template parameter declaration> whose <template parameter type> is TYPE.

b) A <template parameter name> that is contained in <value specification> or <target specification> shall be contained in a <template parameter declaration> whose <template parameter type> is a <data type>.

27) Those <identifier>s that are valid <catalog name>s are implementation-defined.

28) An <exception name> identifies an exception and corresponding <exception value>.

29) If more than one <exception name> uses the same <exception value>, then the <exception name>s shall be considered equivalent.

30) If a <character set name> does not specify a <schema name>, then INFORMATION_SCHEMA is implicit.

31) If a <collation name> does not specify a <schema name>, then INFORMATION_SCHEMA is implicit.

32) If a <translation name> does not specify a <schema name>, then INFORMATION_SCHEMA is implicit.

33) The <data type> of <SQL-server name>, <connection name>, and <user name> shall be character string with an implementation-defined character set and shall have an octet length of 128 octets or less.

34) If a <form-of-use conversion name> does not specify a <schema name>, then INFORMATION_SCHEMA is implicit; otherwise, INFORMATION_SCHEMA shall be specified.

Access Rules

None.

General Rules

1) A <table name> identifies a table.

2) Within its scope, a <correlation name> identifies a table.

ISO Only—caused by ANSI changes not yet considered by ISO

3) A <local table name> identifies a declared local temporary table.

4) A <column name> identifies a column.

5) A <domain name> identifies a domain.

6) An <authorization identifier> represents an authorization identifier and identifies a set of privileges.

7) An <enumeration name> identifies a component of an <enumerated type>.

8) A <module name> identifies a <module>.

9) A <cursor name> identifies a cursor.

10) A <routine name> identifies a <routine>.

11) A <parameter name> identifies a parameter.

ANSI Only–SQL3

12) An <SQL variable name> identifies an SQL variable.

13) An <external routine name> identifies an external routine.

14) A <trigger name> identifies a trigger.

15) A <constraint name> identifies a table constraint, a domain constraint, or an assertion.

16) A <statement name> identifies a statement prepared by the execution of a <prepare statement>. The scope of a <statement name> is the <module> in which it appears and the current SQL-session.

17) The value of an <extended statement name> identifies a statement prepared by the execution of a <prepare statement>. If a <scope option> of GLOBAL is specified, then the scope of the <extended statement name> is the current SQL-session. If a <scope option> of LOCAL is specified or implicit, then the scope of the statement name is further restricted to the <module> in which the <extended statement name> appears.

18) A <dynamic cursor name> identifies a cursor in an <SQL dynamic statement>.

19) The value of an <extended cursor name> identifies a cursor created by the execution of an <allocate cursor statement>. If a <scope option> of GLOBAL is specified, then the scope of the <extended cursor name> is the current SQL-session. If a <scope option> of LOCAL is specified or implicit, then the scope of the cursor name is further restricted to the <module> in which the <extended cursor name> appears.

20) A <descriptor name> identifies an SQL descriptor area created by the execution of an <allocate descriptor statement>. If a <scope option> of GLOBAL is specified, then the scope of the <descriptor name> is the current SQL-session. If a <scope option> of LOCAL is specified or implicit, then the scope of the <descriptor name> is further restricted to the <module> in which the <descriptor name> appears.

21) A <catalog name> identifies a catalog.

22) A <schema name> identifies a schema.

23) A <collation name> identifies a collating sequence.

24) A <character set name> identifies a character set.

25) A <translation name> identifies a character translation.

26) A <form-of-use conversion name> identifies a form-of-use conversion. All <form-of-use conversion name>s are implementation-defined.

27) A <descriptor name> identifies a descriptor area.

28) A <connection name> identifies an SQL-connection.

29) An <abstract data type name> identifies an abstract data type.

30) A <component name> identifies a component of an abstract data type.

• 1 Rule deleted.

31) A <savepoint name> identifies a savepoint. The scope of a <savepoint name> is the SQL-transaction in which it was defined.

32) A <role name> identifies a role.

33) A <null class name> identifies a defined null class. No <null class name> shall specify GENERAL.

34) If the <form-of-use conversion name> does not contain an explicit a <schema name>, then INFORMATION_SCHEMA is implicit; otherwise, INFORMATION_SCHEMA shall be specified.

35) A <type template name> identifies a type template family.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <extended statement name> or <extended cursor name>.

   b) Conforming Intermediate SQL language shall not contain any explicit <catalog name>, <connection name>, <collation name>, <translation name>, <form-of-use conversion name>, or <qualified local table name>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <domain name>, <SQL statement name>, <dynamic cursor name>, <constraint name>, <descriptor name>, or <character set name>.

   b) An <identifier> shall not specify a <character set specification>.

6 Scalar expressions

6.1 <data type>

Function

Specify a data type.

Format

```
<data type> ::=
        <predefined type>
      | <abstract data type name>
      | <generated type reference>
      | <template parameter name>
```

ANSI Only--SQL3

```
      | <collection type>
```

```
<predefined type> ::=
        <character string type> [ CHARACTER SET <character set specification> ]
      | <national character string type>
      | <bit string type>
      | <numeric type>
      | <enumerated type>
      | <boolean type>
      | <datetime type>
      | <interval type>
      | <object identifier type>

<character string type> ::=
        CHARACTER [ <left paren> <length> <right paren> ]
      | CHAR [ <left paren> <length> <right paren> ]
      | CHARACTER VARYING <left paren> <length> <right paren>
      | CHAR VARYING <left paren> <length> <right paren>
      | VARCHAR <left paren> <length> <right paren>

<national character string type> ::=
        NATIONAL CHARACTER [ <left paren> <length> <right paren> ]
      | NATIONAL CHAR [ <left paren> <length> <right paren> ]
      | NCHAR [ <left paren> <length> <right paren> ]
      | NATIONAL CHARACTER VARYING <left paren> <length> <right paren>
      | NATIONAL CHAR VARYING <left paren> <length> <right paren>
      | NCHAR VARYING <left paren> <length> <right paren>

<bit string type> ::=
        BIT [ <left paren> <length> <right paren> ]
      | BIT VARYING <left paren> <length> <right paren>

<numeric type> ::=
        <exact numeric type>
```

```
       | <approximate numeric type>

<exact numeric type> ::=
         NUMERIC [ <left paren> <precision> [ <comma> <scale> ] <right paren> ]
       | DECIMAL [ <left paren> <precision> [ <comma> <scale> ] <right paren> ]
       | DEC [ <left paren> <precision> [ <comma> <scale> ] <right paren> ]
       | INTEGER
       | INT
       | SMALLINT

<approximate numeric type> ::=
         FLOAT [ <left paren> <precision> <right paren> ]
       | REAL
       | DOUBLE PRECISION

<length> ::= <unsigned integer>

<precision> ::= <unsigned integer>

<scale> ::= <unsigned integer>

<enumerated type> ::= <left paren> <enumeration name list> <right paren>

<enumeration name list> ::=
       <enumeration name> [ { <comma> <enumeration name> }... ]

<boolean type> ::= BOOLEAN

<datetime type> ::=
         DATE
       | TIME [ <left paren> <time precision> <right paren> ] [ WITH TIME ZONE ]
       | TIMESTAMP [ <left paren> <timestamp precision> <right paren> ] [ WITH TIME ZONE ]


<time precision> ::= <time fractional seconds precision>

<timestamp precision> ::= <time fractional seconds precision>

<time fractional seconds precision> ::= <unsigned integer>

<interval type> ::= INTERVAL <interval qualifier>

<object identifier type> ::=
       REF [ <left paren> <abstract data type name> <right paren> ]
```

ANSI Only--SQL3

```
       <collection type> ::=
         <set type>
       | <multiset type>
       | <list type>

<set type> ::= SET <left paren> <data type> <right paren>

<multiset type> ::= MULTISET <left paren> <data type> <right paren>

<list type> ::= LIST <left paren> <data type> <right paren>
```

Syntax Rules

1) CHAR is equivalent to CHARACTER. DEC is equivalent to DECIMAL. INT is equivalent to INTEGER. VARCHAR is equivalent to CHARACTER VARYING. NCHAR is equivalent to NATIONAL CHARACTER.

2) ''NATIONAL CHARACTER'' is equivalent to the corresponding <character string type> with a specification of ''CHARACTER SET *CSN*'', where ''*CSN*'' is an implementation-defined <character set name>.

3) The value of a <length> or a <precision> shall be greater than 0.

4) If <length> is omitted, then a <length> of 1 is implicit.

5) If a <scale> is omitted, then a <scale> of 0 is implicit.

6) If a <precision> is omitted, then an implementation-defined <precision> is implicit.

7) CHARACTER specifies the data type character string.

8) Characters in a character string are numbered beginning with 1.

9) Case:

   a) If VARYING is not specified in <character string type>, then the length in characters of the character string is fixed and is the value of <length>.

   b) If VARYING is specified in <character string type>, then the length in characters of the character string is variable, with a minimum length of 0 and a maximum length of the value of <length>.

   The maximum value of <length> is implementation-defined. <length> shall not be greater than this maximum value.

10) If <character string type> is not contained in a <domain definition> or a <column definition> and CHARACTER SET is not specified, then an implementation-defined <character set specification> is implicit.

    Note: Subclause 11.24, "<domain definition>", and Subclause 11.4, "<column definition>", specify the result when <character string type> is contained in a <domain definition> or <column definition>, respectively.

11) The character set named SQL_TEXT is an implementation-defined character set whose character repertoire is SQL_TEXT.

    Note: The character repertoire SQL_TEXT is defined in Subclause 4.2, "Character strings".

12) BIT specifies the data type bit string.

13) Bits in a bit string are numbered beginning with 1.

14) Case:

   a) If VARYING is not specified in <bit string type>, then the length in bits of the bit string is fixed and is the value of <length>.

   b) If VARYING is specified in <bit string type>, then the length in bits of the string is variable, with a minimum length of 0 and a maximum length of the value of <length>.

The maximum value of <length> is implementation-defined. <length> shall not be greater than this maximum value.

15) The <scale> of an <exact numeric type> shall not be greater than the <precision> of the <exact numeric type>.

16) For the <exact numeric type>s DECIMAL and NUMERIC:

a) The maximum value of <precision> is implementation-defined. <precision> shall not be greater than this value.

b) The maximum value of <scale> is implementation-defined. <scale> shall not be greater than this maximum value.

17) NUMERIC specifies the data type exact numeric, with the decimal precision and scale specified by the <precision> and <scale>.

18) DECIMAL specifies the data type exact numeric, with the decimal scale specified by the <scale> and the implementation-defined decimal precision equal to or greater than the value of the specified <precision>.

19) INTEGER specifies the data type exact numeric, with binary or decimal precision and scale of 0. The choice of binary versus decimal precision is implementation-defined, but shall be the same as SMALLINT.

20) SMALLINT specifies the data type exact numeric, with scale of 0 and binary or decimal precision. The choice of binary versus decimal precision is implementation-defined, but shall be the same as INTEGER. The precision of SMALLINT shall be less than or equal to the precision of INTEGER.

21) FLOAT specifies the data type approximate numeric, with binary precision equal to or greater than the value of the specified <precision>. The maximum value of <precision> is implementation-defined. <precision> shall not be greater than this value.

22) REAL specifies the data type approximate numeric, with implementation-defined precision.

23) DOUBLE PRECISION specifies the data type approximate numeric, with implementation-defined precision that is greater than the implementation-defined precision of REAL.

24) For the <approximate numeric type>s FLOAT, REAL, and DOUBLE PRECISION, the maximum and minimum values of the exponent are implementation-defined.

25) An <enumerated type> shall be contained in a <domain definition>.

26) In an <enumerated type>, each <enumeration name> shall be different from any other <enumeration name> of the <enumerated type>.

27) An <enumerated type> defines an ordering of <enumeration name>s. Each <enumeration name> yields a different enumeration value. The predefined order relation between enumeration values follows the order of corresponding position numbers. The position number of the value of the first listed <enumeration name> is 0. The position number of each other <enumeration name> is 1 more than the position number of its predecessor in the list.

28) If <time precision> is not specified, then 0 is implicit. If <timestamp precision> is not specified, then 6 is implicit.

29) The maximum value of <time precision> and the maximum value of <timestamp precision> shall be the same implementation-defined value that is not less than 6. The values of <time precision> and <timestamp precision> shall not be greater than that maximum value.

30) The length of a DATE is 10 positions. The length of a TIME is 8 positions plus the <time fractional seconds precision>, plus 1 position if the <time fractional seconds precision> is greater than 0. The length of a TIME WITH TIME ZONE is 14 positions plus the <time fractional seconds precision> plus 1 position if the <time fractional seconds precision> is greater than 0. The length of a TIMESTAMP is 19 positions plus the <time fractional seconds precision>, plus 1 position if the <time fractional seconds precision> is greater than 0. The length of a TIMESTAMP WITH TIME ZONE is 25 positions plus the <time fractional seconds precision> plus 1 position if the <time fractional seconds precision> is greater than 0.

31) If an <interval qualifier> in an <interval type> includes no fields other than YEAR and MONTH, then the <interval type> is a year-month interval. If an <interval qualifier> in an <interval type> includes any fields other than YEAR or MONTH, then the <interval type> is a day-time interval.

32) The *i*-th value of an interval data type corresponds to the *i*-th <datetime field>.

33) Within the non-null values of a <datetime type>, the value of the time zone interval shall be in the range −12:59 to +13:00.

Note: The range for time zone intervals is larger than many readers might expect because it is governed by political decisions in governmental bodies rather than by any natural law.

34) If <data type>
ISO immediately
contains <abstract data type name>, then there shall exist an abstract data type descriptor specifying that <abstract data type name>.

35) If <data type> is an <object identifier type> containing an <abstract data type name>, then there shall exist an abstract data type descriptor specifying that <abstract data type name> and also specifying WITH OID.

ANSI Only–SQL3

36) <set type> specifies the set data type.

37) <multiset type> specifies the multiset data type.

38) <list type> specifies the list data type.

Access Rules

1) The applicable privileges shall include the USAGE privilege on the abstract data type identified by <abstract data type name>.

Note: The applicable privileges for an <abstract data type name> are defined in Subclause 10.5, "<privileges>".

General Rules

1) If any specification or operation attempts to cause an item of a character type to contain a character that is not a member of the character repertoire associated with the character item, then an exception condition is raised: *data exception—character not in repertoire*.

2) For a <datetime type>,

   Case:

   a) If DATE is specified, then the data type contains the <datetime field>s years, months, and days.

   b) If TIME is specified, then the data type contains the <datetime field>s hours, minutes, and seconds.

   c) If TIMESTAMP is specified, then the data type contains the <datetime field>s years, months, days, hours, minutes, and seconds.

3) For a <datetime type>, a <time fractional seconds precision> that is an explicit or implicit <time precision> or <timestamp precision> defines the number of decimal digits following the decimal point in the SECOND <datetime field>.

4) Table 10, "Valid values for fields in datetime items", speifies the constraints on the values of the <datetime field>s in a datetime data type.

Table 10—Valid values for fields in datetime items

Keyword	Valid values of datetime fields
YEAR	0001 to 9999
MONTH	01 to 12
DAY	Within the range 1 to 31, but further constrained by the value of MONTH and YEAR fields, according to the rules for well-formed dates in the Gregorian calendar.
HOUR	00 to 23
MINUTE	00 to 59
SECOND	00 to 61.9(*N*) where ''9(*N*)'' indicates the number of digits specified by <time fractional seconds precision>.
TIMEZONE_HOUR	00 to 13
TIMEZONE_MINUTE	00 to 59

Note: Datetime data types will allow dates in the Gregorian format to be stored in the date range 0001−01−01 CE through 9999−12−31 CE. The range for SECOND allows for as many as two ''leap seconds''. Interval arithmetic that involves leap seconds or discontinuities in calendars will produce implementation-defined results.

5) If WITH TIME ZONE is not specified, then the time zone displacement of the datetime data type is effectively the current default time zone displacement of the SQL-session.

6) If any specification or operation attempts to cause an item of type datetime to take a value where the <datetime field> YEAR has the value greater than 9999 or less than 1, then an exception condition is raised: *data exception—invalid datetime format*.

7) The values of the <datetime field>s within an interval data type are constrained as follows:

   a) The value corresponding to the first <datetime field> is an integer with at most *N* digits, where *N* is the <interval leading field precision>.

   b) Table 11, "Valid values for fields in INTERVAL items", specifies the constraints for the other <datetime field>s in the interval data type.

Table 11—Valid values for fields in INTERVAL items

Keyword	Valid values of INTERVAL fields
MONTH	0 to 11
HOUR	0 to 23
MINUTE	0 to 59
SECOND	0 to 59.9(*N*) where ''9(*N*)'' indicates the number of digits specified by <interval fractional seconds precision> in the <interval qualifier>.

8) An item of type interval can contain positive or negative intervals.

ANSI Only–SQL3

9) For any <data type> *DT*, the type SET(*DT*) is a subtype of the type MULTISET(*DT*).

10) For any <data type> *DT*, the type LIST(*DT*) is a subtype of the type MULTISET(*DT*).

11) If the <data type> is a <collection type>, then a collection type descriptor is created. The collection type descriptor includes the name of the data type of elements and an indication of whether the collection is a set, multiset, or list.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <data type> shall not be an
   ANSI <abstract type name>, <generated type reference>, or <template parameter name>.
   ISO <object identifier type>.

   b) A <predefined type> shall not be an <enumerated type>, <boolean type>, or <object identifier type>.

ANSI Only–SQL3

   c) A <data type> shall not be a <collection type>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <datetime type> shall not specify a <time precision> or <timestamp precision>.

   b) A <data type> shall not be a <bit string type>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <character string type> shall not specify VARYING or VARCHAR.

   b) A <data type> shall not be a <datetime type> or an <interval type>.

   c) A <data type> shall not be a <national character string type> nor specify CHARACTER SET.

6.2 <value specification> and <target specification>

Function

Specify one or more values, parameters, or variables.

Format

```
<value specification> ::=
        <literal>
      | <general value specification>

<unsigned value specification> ::=
        <unsigned literal>
      | <general value specification>

<general value specification> ::=
```

ANSI Only--SQL3

```
        <item reference>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
        <parameter specification>
      | <variable specification>
```

```
      | USER
      | CURRENT_USER
      | SESSION_USER
      | SYSTEM_USER
      | VALUE
      | <function invocation>
      | <attribute reference>
      | <dynamic parameter specification>
      | <template parameter name>

<function invocation> ::= <routine invocation>

<simple value specification> ::=
```

ANSI Only--SQL3

```
        <item reference>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
        <parameter name>
      | <embedded variable name>
      | <SQL variable name>
```

```
      | <literal>
```

```
<target specification> ::=
```

ANSI Only--SQL3

```
  <item reference>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
    <parameter specification>
  | <variable specification>
  | <dynamic parameter specification>
  | <template parameter name>

<simple target specification> ::=
```

ANSI Only--SQL3

```
    <item reference>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
    <parameter name>
  | <embedded variable name>

<parameter specification> ::=
    <parameter name> [ <indicator parameter> ]

<indicator parameter> ::=
    [ INDICATOR ] <parameter name>

<variable specification> ::=
    { <embedded variable name> | <SQL variable name> } [ <indicator variable> ]

<indicator variable> ::=
    [ INDICATOR ] { <embedded variable name> | <SQL variable name> }

<dynamic parameter specification> ::= <question mark>
```

Syntax Rules

ISO Only—caused by ANSI changes not yet considered by ISO

1) The data type of an <indicator parameter> or <indicator variable> shall be exact numeric with a scale of 0.

ANSI Only–SQL3

2) Each <dynamic parameter specification> shall be contained in a <preparable statement> that is dynamically prepared in the current SQL-session through the execution of a <prepare statement> or an <execute immediate statement>.

3) An <item reference> that is a <simple value specification> or <simple target specification> shall be a host parameter reference or an <embedded variable specification>.

Note: ''Host parameter reference'' is defined in Subclause 6.3, "<item reference>".

ISO Only—caused by ANSI changes not yet considered by ISO

4) Each <parameter name> and <SQL variable name> shall be contained in a <module>. Each <embedded variable name> shall be contained in an <embedded SQL statement>. Each <dynamic parameter specification> shall be contained in a <preparable statement> that is dynamically prepared in the current SQL-session through the execution of a <prepare statement> or an <execute immediate statement>.

5) The data type of USER is fixed-length character string of implementation-defined length. The character set of the character string is the implementation-defined character set that is the default for character string data types that do not specify a CHARACTER SET.

ANSI Only—caused by ISO changes not yet considered by ANSI

6) If USER is specified, then CURRENT_USER is implicit.

7) The data type of CURRENT_USER, SESSION_USER, and SYSTEM_USER is character string. Whether the character string is fixed length or variable length, and its length if it is fixed length or maximum length if it is variable length, are implementation-defined. The character set of the character string is SQL_TEXT. The null class of the character string is the general null class.

ANSI Only–SQL3

8) The <value specification> or <unsigned value specification> VALUE shall be contained either in a <domain constraint> or in an <abstract data type body>.

Case:

a) If VALUE is contained in a <domain constraint>, then the data type of an instance of VALUE is the data type of the domain to which that domain constraint belongs. The null class of an instance of VALUE is the null class of the domain definition to which that domain constraint belongs.

b) If VALUE is contained in an <abstract data type body> *ADTB*, then the data type of an instance of VALUE is

Case:

i) If *ADTB* is contained in an <abstract data type definition>, then the abstract data type of which *ADTB* is the body; or

ii) If *ADTB* is contained in a <type template definition>, then :GEN_TYPE.

ISO Only—caused by ANSI changes not yet considered by ISO

9) The <value specification> or <unsigned value specification> VALUE shall be contained in a <domain constraint>. The data type of an instance of VALUE is the data type of the domain to which that domain constraint belongs. The null class of an instance of VALUE is the null class of the <domain definition> containing that <domain constraint>.

10) If the data type of the <value specification> or <unsigned value specification> is character string, then the <value specification> or <unsigned value specification> has the *Coercible* coercibility attribute, and the collating sequence is determined by Subclause 4.2.3, "Rules determining collating sequence usage".

11) The null class of
ANSI a parameter reference
ISO <parameter specification>
is the general null class. The null class of <variable specification> is the general null class. The null class of <dynamic parameter specification> is the general null class.

12) A <function invocation> shall contain a <routine invocation> whose candidate <routine>s are functions.

Note: The *candidate <routine>s* of a <routine invocation> are defined in Subclause 10.3, "<routine invocation>".

Access Rules

None.

General Rules

1) A <value specification> or <unsigned value specification> specifies a value that is not selected from a table.

ISO Only—caused by ANSI changes not yet considered by ISO

2) A <parameter specification> identifies a parameter or a parameter and an indicator parameter in a <module>.

3) A <variable specification> identifies a host variable or a host variable and an indicator variable.

4) A <dynamic parameter specification> identifies a parameter used by a dynamically prepared statement.

5) A <target specification> specifies a parameter or variable that can be assigned a value.

ISO Only—caused by ANSI changes not yet considered by ISO

6) If a <parameter specification> contains an <indicator parameter> and the value of the indicator parameter is negative, then the value specified by the <parameter specification> is a null value. Otherwise, the value specified by a <parameter specification> is the value of the parameter identified by the <parameter name>.

7) If a <variable specification> contains an <indicator variable> and the value of the indicator variable is negative, then the value specified by the <variable specification> is a null value. Otherwise, the value specified by a <variable specification> is the value of the variable identified by the <embedded variable name>.

8) If an <indicator parameter> is negative, then

   Case:

   a) If the value of the <indicator parameter> is −1, the null value is the general null value.

   b) Otherwise, the null value is the null state whose position number within the null class of the appropriate <value specification> or <unsigned value specification> or <parameter specification> is given by the absolute value of 1 less than value of the <indicator parameter>.

9) The value specified by a <literal> is the value represented by that <literal>.

ISO Only–SQL3

10) The value specified by USER is the value of the current <authorization identifier>. If this value contains characters not in the character repertoire of the data type of USER, then an exception condition is raised: *data exception—unrepresentable value for USER*.

11) The value specified by CURRENT_USER is the value of the current <authorization identifier>.

12) The value specified by SESSION_USER is the value of the SQL-session <authorization identifier>.

13) The value specified by SYSTEM_USER is equal to an implementation-defined string that represents the operating system user who executed the <module> that contains the SQL-statement whose execution caused the SYSTEM_USER <general value specification> to be evaluated.

14) A <simple value specification> specifies a <value specification> or <unsigned value specification> that is not null and does not have an associated <indicator parameter> or <indicator variable>.

15) A <simple target specification> specifies a parameter or variable that can be assigned a value that is not null.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   ANSI Only–SQL3

   None.

   ISO Only—caused by ANSI changes not yet considered by ISO

   a) In Intermediate SQL, the specific data type of <indicator parameter>s and <indicator variable>s shall be the same implementation-defined data type.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <general value specification> shall not be a <dynamic parameter specification>.

   b) A <general value specification> shall not specify VALUE.

   c) A <general value specification> shall not specify CURRENT_USER, SYSTEM_USER, or SESSION_USER.

      Note: Although CURRENT_USER and USER are semantically the same, in Entry SQL, CURRENT_USER must be specified as USER.

ANSI Only–SQL3

6.3 <item reference>

Function

Reference a parameter or an SQL variable.

Format

```
<item reference> ::=
          <parameter name> [ <indicator parameter> ]
        | <SQL variable name>
        | <embedded variable specification>

<indicator parameter> ::= [ INDICATOR ] <parameter name>

<embedded variable specification> ::=
       <embedded variable name> [ <indicator variable> ]

<indicator variable> ::= [ INDICATOR ] <embedded variable name>
```

Syntax Rules

1) Let *IR* be an <item reference>.

   Case:

   a) If *IR* is contained in a <declare cursor>, then *IR* shall be defined in a <parameter declaration> in the <routine> in the containing <module> that contains an <open statement> that specifies the <cursor name> of the <declare cursor>. *IR* denotes that parameter.

   b) Otherwise, *IR* shall be contained in one or more <compound statement>s or <routine>s that contain an <SQL variable declaration> or <parameter declaration> whose <parameter name> or <SQL variable name> is *IR*. *IR* denotes that parameter or SQL variable in the innermost such <routine> or <compound statement>.

2) If *IR* denotes an SQL variable, then *IR* is an ''SQL variable reference''. If *IR* denotes a parameter, then *IR* is a ''parameter reference''. If *IR* is a parameter reference that denotes an externally-supplied parameter, then *IR* is a ''host parameter reference''. If *IR* is a parameter reference that denotes an SQL-supplied parameter, then *IR* is an ''SQL parameter reference''.

3) An <indicator parameter> shall be specified only if the <item reference> is a host parameter reference. *IR* and the <parameter name> of the <indicator parameter> shall identify parameters of the same <routine>. The data type of the <indicator parameter> shall be exact numeric with a scale of 0.

4) Each <embedded variable name> shall be contained in an <embedded SQL statement>.

Access Rules

None.

General Rules

1) If a host parameter reference contains an <indicator parameter> and the value of the indicator parameter is negative, then the value specified by the host parameter reference is a null value. Otherwise, the value specified by a host parameter reference is the value of the parameter.

2) If an <embedded variable specification> contains an <indicator variable> and the value of the indicator variable is negative, then the value specified by the <embedded variable specification> is a null value. Otherwise, the value specified by the <embedded variable specification> is the value identified by the <embedded variable name>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) In Intermediate SQL, the specific data type of <indicator parameter>s and <indicator variable>s shall be the same implementation-defined data type.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

6.4 <attribute reference>

Function

Access an attribute in an abstract data type instance.

Format

```
<attribute reference> ::=
       <value specification> <period> <column name>
```

Syntax Rules

1) The datatype of <value specification> shall be an abstract data type or shall be an object identifier type that is qualified by an abstract data type name. Let *ADT* be the abstract data type.

2) <attribute reference> shall be contained in either the <abstract data type definition> that defines *ADT* or in an <routine> whose <routine name> is specified in the <operator name list> of that <abstract data type definition>.

3) If <attribute reference> appears as the <assignment target> of an <assignment statement>, then:

   a) The <column name> shall not be OID; and

   b) If the <attribute reference> identifies a virtual attribute, then there shall be a set function *SF* associated with the virtual attribute.

   c) <column name> shall be the name of a column specified in *ADT*.
   ANSI Let *C* denote the attribute of the ADT instance identified by <column name>.
   ISO Let *i* denote the ordinal position of this column in the ADT descriptor.

Access Rules

None.

General Rules

ANSI Only–SQL3

1) Let *I* denote the abstract data type instance.

2) If <attribute reference> appears as the <value expression> of an <assignment statement>, then

Case:

a) If the attribute is a stored attribute, then the value of <attribute reference> is the value of the
ANSI *C*.
ISO *i*-th component of *I*.

b) If the attribute is a virtual attribute, then let *GF* be the get function associated with the virtual attribute; the value of <attribute reference> is the result of performing an SQL function invocation of *GF*(*I*).

3) If <attribute reference> appears as the <assignment target> of an <assignment statement>, then

Case:

a) If the attribute is a stored attribute, then the <attribute reference> identifies the
ANSI *C*.
ISO *i*-th component of *I*.

b) If the attribute is a virtual attribute, then the <attribute reference> identifies the instance *I* and the set function *SF*.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) Conforming Full SQL language shall contain no <attribute reference>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

6.5 <table reference>

Function

Reference a table.

Format

```
<table reference> ::=
         <table name> [ [ AS ] <correlation name>
             [ <left paren> <derived column list> <right paren> ] ]
       | <derived table> [ AS ] <correlation name>
             [ <left paren> <derived column list> <right paren> ]
       | <joined table>

<derived table> ::= <table subquery>

<derived column list> ::= <column name list>

<column name list> ::=
       <column name> [ { <comma> <column name> }... ]
```

Syntax Rules

1) A <correlation name> immediately contained in a <table reference> *TR* is *exposed* by *TR*. A <table name> immediately contained in a <table reference> *TR* is *exposed* by *TR* if and only if *TR* does not specify a <correlation name>.

2) Case:

   a) If a <table reference> *TR* is contained in a <from clause> *FC* with no intervening <derived table>, then the *scope clause SC* of *TR* is the <select statement: single row> or innermost <query specification> that contains *FC*. The scope of the exposed <correlation name> or exposed <table name> of *TR* is the <select list>, <where clause>, <group by clause>, and <having clause> of *SC*, together with the <join condition> of all <joined table>s contained in *SC* that contains *TR*.

   b) Otherwise, the *scope clause SC* of *TR* is the outermost <joined table> that contains *TR* with no intervening <derived table>. The scope of the exposed <correlation name> or exposed <table name> of *TR* is the <join condition> of *SC* and of all <joined table>s contained in *SC* that contain *TR*.

3) A <table name> that is exposed by a <table reference> *TR* shall not be the same as any other <table name> that is exposed by a <table reference> with the same scope clause as *TR*.

4) A <correlation name> that is exposed by a <table reference> *TR* shall not be the same as any other <correlation name> that is exposed by a <table reference> with the same scope clause as *TR* and shall not be the same as the <qualified identifier> of any <table name> that is exposed by a <table reference> with the same scope clause as *TR*.

5) A <table name> immediately contained in a <table reference> *TR* has a scope clause and scope defined by that <table reference> if and only if the <table name> is exposed by *TR*.

6) The same <column name> shall not be specified more than once in a <derived column list>.

7) If a <derived column list> is specified in a <table reference>, then the number of <column name>s in the <derived column list> shall be the same as the degree of the table specified by the <derived table> or the <table name> of that <table reference>, and the name of the *i*-th column of that <derived table> or the effective name of the *i*-th column of that <table name> is the *i*-th <column name> in that <derived column list>.

ANSI Only–SQL3

8) Let *QE* be the <query expression> simply contained in the <subquery> of the <table subquery> of the <derived table>.

A <derived table> is an *updatable derived table* if and only if at least one of the following conditions hold:

a) *QE* is updatable;

b) The <derived table> is the subject of a <trigger definition> specifying a <trigger action time> of INSTEAD OF and a <trigger event> of UPDATE for all columns of the <derived table>; or

c) The <derived table> is the subject of one or more <trigger definition>s that specify a <trigger action time> INSTEAD OF and between them specifying a <trigger event> of UPDATE for any column of the <derived table> for which the corresponding column of *QE* is not updatable.

ISO Only—caused by ANSI changes not yet considered by ISO

9) A <derived table> is an *updatable derived table* if and only if the <query expression> simply contained in the <subquery> of the <table subquery> of the <derived table> is updatable.

Access Rules

1) Let *T* be the table identified by the <table name> immediately contained in <table reference>. If the <table reference> is contained in any of:

a) a <query expression> simply contained in a <cursor specification>, a <view definition>, a <direct select statement: multiple rows>, or an <insert statement>; or

b) a <table expression> or <select list> immediately contained in a <select statement: single row>; or

c) a <search condition> immediately contained in a <delete statement: searched> or an <update statement: searched>; or

d) a <value expression> immediately contained in an <update source>,

then the applicable privileges shall include SELECT for
ANSI *T* or for
at least one column of *T*.

General Rules

1) The <correlation name> or exposed <table name> contained in a <table reference> defines that <correlation name> or <table name> to be an identifier of the table identified by the <table name> or <derived table> of that <table reference>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <table reference> shall not be a <derived table>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <table reference> shall not be a <joined table>.

   b) The optional <key word> AS shall not be specified.

   c) <derived column list> shall not be specified.

6.6 <column reference> and <object reference>

Function

Reference a column or an object.

Format

```
<column reference> ::= [ <qualifier> <period> ] <column name>

<object reference> ::= <qualifier>

<qualifier> ::=
         <table name>
       | <correlation name>
```

Syntax Rules

1) Let *CR* be the <column reference>, let *CN* be the <column name> contained in *CR*, and let *C* be the column identified by *CN*.

2) If *CR* contains a <qualifier> *Q*, then *CR* shall appear within the scope of one or more <table name>s or <correlation name>s that are equal to *Q*. If there is more than one such <table name> or <correlation name>, then the one with the most local scope is specified. Let *T* be the table associated with *Q*.

   a) *T* shall include a column whose <column name> is *CN*.

   b) If *T* is a <table reference> in a <joined table> *J*, then *CN* shall not be a common column name in *J*.

      Note: *common column name* is defined in Subclause 7.8, "<joined table>".

3) If *CR* does not contain a <qualifier>, then *CR* shall be contained within the scope of one or more <table name>s or <correlation name>s whose associated tables include a column whose <column name> is *CN*. Let the phrase *possible qualifiers* denote those <table name>s and <correlation name>s.

   a) Case:

      i) If the most local scope contains exactly one possible qualifier, then the qualifier *Q* equivalent to that unique <table name> or <correlation name> is implicit.

      ii) If there is more than one possible qualifier with most local scope, then:

         1) Each possible qualifier shall be a <table name> or a <correlation name> of a <table reference> that is directly contained in a <joined table> *J*.

         2) *CN* shall be a common column name in *J*.

            Note: *common column name* is defined in Subclause 7.8, "<joined table>".

         3) The implicit qualifier *Q* is implementation-dependent. The scope of *Q* is that which *Q* would have had if *J* had been replaced by the <table reference>:

            (*J*) AS *Q*

b) Let *T* be the table associated with *Q*.

4) The data type of *CR* is the data type of column *C* of *T*. *CN* shall uniquely identify a column of *T*.

5) If the data type of *CR* is character string, then *CR* has the *Implicit* coercibility attribute and its collating sequence is the default collating sequence for column *C* of *T*.

6) If the data type of *CR* is TIME or TIMESTAMP, then the implicit time zone of the data is the current default time zone for the SQL-session.

7) If the data type of *CR* is TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE, then the time zone of the data is the time zone represented in the value of *CR*.

8) If *CR* is contained in a <table expression> *TE* or a <there is predicate> *TE* or a <quantified predicate> *TE* immediately containing an <existential clause> or a <universal clause> and the scope of *Q* is some <SQL procedure statement>, <recursive union>, <trigger definition>, <quantified predicate>, or <table reference> that contains *TE*, then *CR* is an *outer reference* to the table associated with *Q*.

9) Let *CR* be the <column reference> and let *C* be the column identified by *CR*. *C* is an underlying column of *CR*. If *C* is a <derived column>, then every underlying column of *C* is an underlying column of *CR*.

Note: The underlying columns of a <derived column> are defined in Subclause 7.12, "<query specification>".

10) An <object reference> *Q* shall appear within the scope of one or more <table name>s or <correlation name>s that is equal to *Q*. If there is more than one such <table name> or <correlation name>, then the one with the most local scope is specified. Let *T* be the table associated with that <table name> or <correlation name>.

11) The data type of *Q* is the <abstract data type name> of the abstract data type defined for table *T*.

12) If *Q* is contained in a <table expression> *TE* or a <there is predicate> *TE* or a <quantified predicate> *TE* immediately containing an <existential clause> or a <universal clause> and the scope clause of *Q* is some <SQL procedure statement>, <recursive union>, <trigger definition>, <quantified predicate>, or <table reference> that contains *TE*, then *Q* is an *outer reference* to the table associated with *Q*.

Access Rules

1) The applicable privileges shall include SELECT for *T* if *CR* is contained in any of:

   a) a <query expression> simply contained in a <cursor specification>, a <view definition>, an <insert statement>, a <direct select statement: multiple rows>, a <temporary view definition>; or

   b) a <sort specification list> contained in a <cursor specification> or a <direct select statement: multiple rows>; or

   c) a <table expression> or <there is predicate> immediately contained in a <select statement: single row>; or

   d) a <search condition> immediately contained in a <delete statement: searched> or an <update statement: searched>; or

e) a <select list> immediately contained in a <select statement: single row>; or

f) a <query specification> contained in a <dynamic single row select statement>; or

g) a <value expression> immediately contained in an <update source>.

General Rules

1) The <column reference> *Q.CN* references column *C* in a given row of *T*.

2) The <object reference> *Q* references a given object of *T*.

ANSI Only–SQL3

If the row type of *T* is a predefined type, theis is equivalent to the <column reference> ''*T*.CONTENTS''.

3) If the data type of *CR* is TIME, TIMESTAMP, TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE, then let *TZ* be an INTERVAL HOUR TO MINUTE containing the value of the time zone displacement associated with *CR*. The value of *CR*, normalized to UTC, is effectively computed as

CR + TZ.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions;

None.

3) The following restrictions apply for Entry SQL;

None.

6.7 <set function specification>

Function

Specify a value derived by the application of a function to an argument.

Format

```
<set function specification> ::=
         COUNT <left paren> <asterisk> <right paren>
       | <general set function>

<general set function> ::=
         <set function type>
             <left paren> [ <set quantifier> ] <value expression> <right paren> ]

<set function type> ::=
       AVG | MAX | MIN | SUM | COUNT

<set quantifier> ::= DISTINCT | ALL
```

Syntax Rules

1) If <set quantifier> is not specified, then ALL is implicit.

2) The argument of COUNT(*) and the argument source of a <general set function> is a table or a group of a grouped table as <general set function> is a table or a group of a grouped table as specified in Subclause 7.11, "<having clause>", and Subclause 7.12, "<query specification>".

   Note: ''argument source'' is defined in Subclause 7.11, "<having clause>".

3) Let *T* be the argument or argument source of a <set function specification>.

4) The <value expression> simply conained in <set function specification> shall not contain a <set function specification> or a <subquery>. If the <value expression> contains a <column reference> that is an outer reference, then that outer reference shall be the only <column reference> contained in the <value expression>.

   Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

5) If a <set function specification> contains a <column reference> that is an outer reference, then the <set function specification> shall be contained in either:

   a) a <select list>, or

   b) a <subquery> of a <having clause>, in which case the scope of the explicit or implicit <qualifier> of the <column reference> shall be a <table reference> that is directly contained in the <table expression> that directly contains the <having clause>.

   Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

6) If the <set function specification> specifies a <set function type> that is MAX or MIN and the <value expression> contained in a <set function specification> simply contains a <value specification> whose data type is an abstract data type, then the <abstract data type definition> that defined that abstract data type shall contain a <less-than clause> that does not specify NONE.

ANSI Only–SQL3

7) If the <set function specification> specifies a <set type> that is AVG, MAX, MIN, or SUM, then the data type of <value expression> simply contained in <general set function> shall not be a <collection type>.

8) If the <set function specification> specifies a <set type> that is AVG or SUM, then the <value expression> contained in the <set function specification> shall not simply contain a <value specification> whose data type is an abstract data type.

9) Let *DT* be the data type of the <value expression>.

10) If COUNT is specified, then the data type of the result is exact numeric with implementation-defined precision and scale of 0.

11) If MAX or MIN is specified, then the data type of the result is *DT*.

12) If SUM or AVG is specified, then:

    a) *DT* shall not be character string, bit string, an enumerated type, or datetime.

    b) If SUM is specified and *DT* is exact numeric with scale *S*, then the data type of the result is exact numeric with implementation-defined precision and scale *S*.

    c) If AVG is specified and *DT* is exact numeric, then the data type of the result is exact numeric with implementation-defined precision not less than the precision of *DT* and implementation-defined scale not less than the scale of *DT*.

    d) If *DT* is approximate numeric, then the data type of the result is approximate numeric with implementation-defined precision not less than the precision of *DT*.

    e) If *DT* is interval, then the data type of the result is interval with the same precision as *DT*.

13) If the data type of the result is character string, then the collating sequence and the coercibility attribute are determined as in Subclause 4.2.3, "Rules determining collating sequence usage".

14) The null class of the result is the null class of the values that result from evaluation of <value expression>

Access Rules

None.

General Rules

1) Case:

    a) If COUNT(*) is specified, then the result is the cardinality of *T*.

    b) Otherwise, let *TX* be the single-column table that is the result of applying the <value expression> to each row of *T* and eliminating null values. If one or more null values are eliminated, then a completion condition is raised: *warning—null value eliminated in set function*.

2) If DISTINCT is specified, then let *TXA* be the result of eliminating redundant duplicate values from *TX*, using the comparison rules specified in Subclause 8.2, "<comparison predicate>" to identify the redundant duplicate values. Otherwise, let *TXA* be *TX*.

   Case:

   a) If the <general set function> COUNT is specified, then the result is the cardinality of *TXA*.

   b) If AVG, MAX, MIN, or SUM is specified, then

      Case:

      i) If *TXA* is empty, then the result is the general null value.

      ii) If AVG is specified, then the result is the average of the values in *TXA*.

      iii) If MAX or MIN is specified, then the result is respectively the maximum or minimum value in *TXA*. These results are determined using the comparison rules specified in Subclause 8.2, "<comparison predicate>".

      iv) If SUM is specified, then the result is the sum of the values in *TXA*. If the sum is not within the range of the data type of the result, then an exception condition is raised: *data exception—numeric value out of range*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) If a <general set function> specifies DISTINCT, then the <value expression> shall be a <column reference>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If a <general set function> specifies or implies ALL, then COUNT shall not be specified.

   b) If a <general set function> specifies or implies ALL, then the <value expression> shall include a <column reference> that references a column of *T*.

   c) If the <value expression> contains a <column reference> that is an outer reference, then the <value expression> shall be a <column reference>.

   d) No <column reference> contained in a <set function specification> shall reference a column derived from a <value expression> that generally contains a <set function specification>.

6.8 <numeric value function>

Function

Specify a function yielding a value of type numeric.

Format

```
<numeric value function> ::=
         <position expression>
       | <extract expression>
       | <length expression>

<position expression> ::=
       POSITION <left paren> <character value expression>
           IN <character value expression> <right paren>

<length expression> ::=
         <char length expression>
       | <octet length expression>
       | <bit length expression>

<char length expression> ::=
       { CHAR_LENGTH | CHARACTER_LENGTH }
           <left paren> <string value expression> <right paren>

<octet length expression> ::=
       OCTET_LENGTH <left paren> <string value expression> <right paren>

<bit length expression> ::=
       BIT_LENGTH <left paren> <string value expression> <right paren>

<extract expression> ::=
       EXTRACT <left paren> <extract field>
           FROM <extract source> <right paren>

<extract field> ::=
         <datetime field>
       | <time zone field>

<time zone field> ::=
         TIMEZONE_HOUR
       | TIMEZONE_MINUTE

<extract source> ::=
         <datetime value expression>
       | <interval value expression>
```

Syntax Rules

1) If <position expression> is specified, then the character repertoires of the two <character value expression>s shall be the same.

2) If <position expression> is specified, then the data type of the result is exact numeric with implementation-defined precision and scale 0.

3) If <extract expression> is specified, then

Case:

a) If <extract field> is a <datetime field>, then it shall identify a <datetime field> of the <interval value expression> or <datetime value expression> immediately contained in <extract source>.

b) If <extract field> is a <time zone field>, then the data type of the <extract source> shall be TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE.

4) If <extract expression> is specified, then

Case:

a) If <datetime field> does not specify SECOND, then the data type of the result is exact numeric with implementation-defined precision and scale 0.

b) Otherwise, the data type of the result is exact numeric with implementation-defined precision and scale. The implementation-defined scale shall not be less than the specified or implied <time fractional seconds precision> or <interval fractional seconds precision>, as appropriate, of the SECOND <datetime field> of the <extract source>.

5) If a <length expression> is specified, then the data type of the result is exact numeric with implementation-defined precision and scale 0.

6) If all of the <character value expression>s, <string value expression>s, <datetime value expression>s, and <interval value expression>s that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) If the value of one or more <character value expression>s, <string value expression>s, <datetime value expression>s, and <interval value expression>s that are simply contained in a <numeric value function> are null values, then

Case:

a) If the result has the general null class, then the result of the <numeric value function> is the general null value.

b) Otherwise the result of the <numeric value function> is the minimum of the null values of the operands.

2) If <position expression> is specified, then

Case:

a) If the first <character value expression> has a length of 0, then the result is 1.

b) If the value of the first <character value expression> is equal to an identical-length substring of contiguous characters from the value of the second <character value expression>, then the result is 1 greater than the number of characters within the value of the second <character value expression> preceding the start of the first such substring.

c) Otherwise, the result is 0.

3) If <extract expression> is specified, then

Case: LE>If <extract field> is a <datetime field>, then the result is the value of the datetime field identified by that <datetime field> and has the same sign as the <extract source>.

Note: If the value of the identified <datetime field> is zero or if <extract source> is not an <interval value expression>, then the sign is irrelevant.

a) Otherwise, let *TZ* be the interval value of the implicit or explicit time zone associated with the <datetime value expression>. If <extract field> is TIMEZONE_HOUR, then the result is calculated as

EXTRACT (HOUR FROM *TZ*)

Otherwise, the result is calculated as

EXTRACT (MINUTE FROM *TZ*)

4) If a <char length expression> is specified, then let *S* be the <string value expression>.

Case:

a) If the data type of *S* is character string, then the result is the number of characters in the value of *S*.

b) Otherwise, the result is OCTET_LENGTH(*S*).

5) If an <octet length expression> is specified, then let *S* be the <string value expression>. The result of the <octet length expression> is the smallest integer not less than the quotient of the division (BIT_LENGTH(*S*)/8).

6) If a <bit length expression> is specified, then let *S* be the <string value expression>. The result of the <bit length expression> is the number of bits in the value of *S*.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) A <numeric value function> shall not be a <position expression>.

b) A <numeric value function> shall not contain a <length expression> that is a <bit length expression>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <numeric value function> shall not be a <length expression>.

b) A <numeric value function> shall not be an <extract expression>.

6.9 <string value function>

Function

Specify a function yielding a value of type character string or bit string.

Format

```
<string value function> ::=
         <character value function>
       | <bit value function>

<character value function> ::=
         <character substring function>
       | <regular expression substring function>
       | <fold>
       | <form-of-use conversion>
       | <character translation>
       | <trim function>

<character substring function> ::=
       SUBSTRING <left paren> <character value expression> FROM <start position>
                   [ FOR <string length> ] <right paren>

<regular expression substring function> ::=
       SUBSTRING <left paren> <character value expression> FROM
                   <character value expression> FOR <escape character> <right paren>

<fold> ::= { UPPER | LOWER } <left paren> <character value expression> <right paren>

<form-of-use conversion> ::=
       CONVERT <left paren> <character value expression>
           USING <form-of-use conversion name> <right paren>

<character translation> ::=
       TRANSLATE <left paren> <character value expression>
           USING <translation name> <right paren>

<trim function> ::=
       TRIM <left paren> <trim operands> <right paren>

<trim operands> ::=
       [ [ <trim specification> ] [ <trim character> ] FROM ] <trim source>

<trim source> ::= <character value expression>

<trim specification> ::=
         LEADING
       | TRAILING
       | BOTH

<trim character> ::= <character value expression>

<bit value function> ::=
       <bit substring function>

<bit substring function> ::=
       SUBSTRING <left paren> <bit value expression> FROM <start position>
           [ FOR <string length> ] <right paren>
```

```
<start position> ::= <numeric value expression>

<string length> ::= <numeric value expression>
```

Syntax Rules

1) The data type of a <start position> and <string length> shall be exact numeric with scale 0.

2) If <character substring function> is specified, then:

   a) The data type of the <character substring function> is variable-length character string with maximum length equal to the fixed length or maximum variable length of the <character value expression>. The character repertoire and form-of-use of the <character substring function> are the same as the character repertoire and form-of-use of the <character value expression>.

   b) The collating sequence and the coercibility attribute are determined as specified for monadic operators in Subclause 4.2.3, "Rules determining collating sequence usage", where the first operand of SUBSTRING plays the role of the monadic operand.

3) If <regular expression substring function> is specified, then the data types of the <escape character> and the <character value expression>s of the <regular expression substring function> shall be character string with the same character repertoire; the data type of the <regular expression substring function> shall be character string with the same character repertoire. The value of the <escape character> shall have length 1.

4) If <fold> is specified, then:

   a) The data type of the result of <fold> is the data type of the <character value expression>.

   b) The collating sequence and the coercibility attribute are determined as specified for monadic operators in Subclause 4.2.3, "Rules determining collating sequence usage", where the operand of the <fold> is the monadic operand.

5) If <form-of-use conversion> is specified, then:

   a) A <form-of-use conversion name> shall identify a form-of-use conversion.

   b) The data type of the result is variable-length character string with implementation-defined maximum length. The character set of the result is the same as the character repertoire of the <character value expression> and form-of-use determined by the form-of-use conversion identified by the <form-of-use conversion name>. Let *CR* be that character repertoire. The result has the *Implicit* coercibility attribute and its collating sequence is *X*, where *X* is the default collating sequence of *CR*.

6) If <character translation> is specified, then:

   a) A <translation name> shall identify a character translation.

   b) The data type of the <character translation> is variable-length character string with implementation-defined maximum length and character repertoire equal to the character repertoire of the target character set of the translation. Let *CR* be that character repertoire. The result has the *Implicit* coercibility attribute and its collating sequence is *X*, where *X* is the default collating sequence of *CR*.

7) If <trim function> is specified, then;

   a) If FROM is specified, then either <trim specification> or <trim character> or both shall be specified.

   b) If <trim specification> is not specified, then BOTH is implicit.

   c) If <trim character> is not specified, then ' ' is implicit.

   d) If

      TRIM (*SRC*)

      is specified, then

      TRIM (BOTH ' ' FROM *SRC*)

      is implicit.

   e) The data type of the <trim function> is variable-length character string with maximum length equal to the fixed length or maximum variable length of the <trim source>.

   f) If a <trim character> is specified, then <trim character> and <trim source> shall be comparable.

   g) The character repertoire and form-of-use of the <trim function> are the same as those of the <trim source>.

   h) The collating sequence and the coercibility attribute are determined as specified for monadic operators in Subclause 4.2.3, "Rules determining collating sequence usage", where the <trim source> of TRIM plays the role of the monadic operand.

8) If <bit substring function> is specified, then the data type of the <bit substring function> is variable-length bit string with maximum length equal to the fixed length or maximum variable length of the <bit value expression>.

9) The null class of a <character value function> is determined as follows:

   Case:

   a) In a <character substring function>,

      Case:

      i) If those of <character value expression>, <start position> and <string length> that do not have the general null class have the same null class, then the result has that null class.

      ii) Otherwise, the result has the general null class.

   b) In a <regular expression substring function>,

      Case:

      i) If those out of <character value expression>s and <escape character> that do not have the general null class have the same null class, then the result has that null class.

      ii) Otherwise, the result has the general null class.

Access Rules

1) The applicable privileges shall include USAGE for every <translation name> contained in the <string value expression>.

General Rules

1) If <character substring function> is specified, then:

   a) Let *C* be the value of the <character value expression>, let *LC* be the length of *C*, and let *S* be the value of the <start position>.

   b) If <string length> is specified, then let *L* be the value of <string length> and let *E* be *S* +*L*. Otherwise, let *E* be the larger of *LC* + 1 and *S*.

   c) If either *C*, *S*, or *L* is a null value, then the result of the <character substring function> is the general null value.

   d) If *E* is less than *S*, then an exception condition is raised: *data exception—substring error*.

   e) Case:

      i) If *S* is greater than *LC* or if *E* is less than 1, then the result of the <character substring function> is a zero-length string.

      ii) Otherwise,

         1) Let *S1* be the larger of *S* and 1. Let *E1* be the smaller of *E* and *LC*+1. Let *L1* be *E1* −*S1*.

         2) The result of the <character substring function> is a character string containing the *L1* characters of *C* starting at character number *S1* in the same order that the characters appear in *C*.

2) If <regular expression substring function> is specified, then:

   i) Let *C* be the result of the first <character value expression>, let *R* be the result of the second <character value expression>, and let *E* be the result of the <escape character>.

   ii) If one or more of *C*, *R* or *E* is a null value, then,

      Case:

         1) If the result has the general null class, then the result of the <regular expression substring function> is the general null value.

         2) Otherwise the result of the <regular expression substring function> is the minimum of the null values of the operands.

   iii) *R* is divided into three substring parts, *R1*, *R2*, and *R3*, such that *R1*, *R2*, and *R3* are regular expressions and

      '*R*' = '*R1*' || '*E*' || '"' || '*R2*' || '*E*' || '"' || '*R3*'

      is true. If an <escape character> is specified, then

      '*C*' SIMILAR TO '*R1*' || '*R2*' || '*R3*'

      is true.

iv) If

'*C*' SIMILAR TO '*R1*' || '*R2*' || '*R3*' ESCAPE '*E*'

is not true, then the result of the <regular expression substring function> is the general null value.

v) If *C* is not a null value, then the result *S* of the <regular expression substring function> satisfies the following conditions:

1) *S* is a substring of *C* such that there are substrings *S1* and *S2* and

'*C*' = '*S1*' || '*S*' || '*S2*'

is true.

2) '*S1*' SIMILAR TO '*R1*' ESCAPE '*E*'

is true and for any substring *S* of *S1*,

'*S*' SIMILAR TO '*R1*' ESCAPE '*E*'

is false.

3) '*S*' SIMILAR TO '*R2*' ESCAPE '*E*'

is true and for any substring *S* of *S*,

'*S*' SIMILAR TO '*R2*' ESCAPE '*E*'

is false.

3) If <fold> is specified, then:

a) Let *S* be the value of the <character value expression>.

b) If *S* is a null value, then the result of the <fold> is the general null value.

c) Case:

i) If UPPER is specified, then the result of the <fold> is a copy of *S* in which every <simple Latin lower case letter> that has a corresponding <simple Latin upper case letter> in the character repertoire of *S* is replaced by that <simple Latin upper case letter>.

ii) If LOWER is specified, then the result of the <fold> is a copy of *S* in which every <simple Latin upper case letter> that has a corresponding <simple Latin lower case letter> in the character repertoire of *S* is replaced by that <simple Latin lower case letter>.

4) If a <character translation> is specified, then

Case:

a) If the value of <character value expression> is a null value, then the result of the <character translation> is the general null value.

b) Otherwise, the value of the <character translation> is the value of the <character value expression> after translation to the character repertoire of the target character set of the translation.

5) If a <form-of-use conversion> is specified, then

Case:

a) If the value of <character value expression> is a null value, then the result of the <form-of-use conversion> is the general null value.

b) Otherwise, the value of the <form-of-use conversion> is the value of the <character value expression> after the application of the form-of-use conversion specified by <form-of-use conversion name>.

6) If <trim function> is specified, then:

a) Let *S* be the value of the <trim source>.

b) If <trim character> is specified, then let *SC* be the value of <trim character>; otherwise, let *SC* be <space>.

c) If either *S* or *SC* is a null value, then the result of the <trim function> is the general null value.

d) If the length in characters of *SC* is not 1, then an exception condition is raised: *data exception—trim error*.

e) Case:

i) If BOTH is specified or if no <trim specification> is specified, then the result of the <trim function> is the value of *S* with any leading or trailing characters equal to *SC* removed.

ii) If TRAILING is specified, then the result of the <trim function> is the value of *S* with any trailing characters equal to *SC* removed.

iii) If LEADING is specified, then the result of the <trim function> is the value of *S* with any leading characters equal to *SC* removed.

7) If <bit substring function> is specified, then:

a) Let *B* be the value of the <bit value expression>, let *LB* be the length in bits of *B*, and let *S* be the value of the <start position>.

b) If <string length> is specified, then let *L* be the value of <string length> and let *E* be *S* +*L*. Otherwise, let *E* be the larger of *LB* + 1 and *S*.

c) If either *B*, *S*, or *L* is a null value, then the result of the <bit substring function> is the general null value.

d) If *E* is less than *S*, then an exception condition is raised: *data exception—substring error*.

e) Case:

i) If *S* is greater than *LB* or if *E* is less than 1, then the result of the <bit substring function> is a zero-length string.

ii) Otherwise,

1) Let *S1* be the larger of *S* and 1. Let *E1* be the smaller of *E* and *LB*+1. Let *L1* be *E1* −*S1*.

2) The result of the <bit substring function> is a bit string containing *L1* bits of *B* starting at bit number *S1* in the same order that the bits appear in *B*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <string value function> shall not be a <regular expression substring function>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <character value function> shall not be a <fold>.

   b) Conforming Intermediate SQL language shall contain no <character translation>.

   c) Conforming Intermediate SQL language shall contain no <form-of-use conversion>.

   d) Conforming Intermediate SQL language shall contain no <bit value function>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <character value function> shall not be a <character substring function>.

   b) A <character value function> shall not be a <trim function>.

6.10 <datetime value function>

Function

Specify a function yielding a value of type datetime.

Format

```
<datetime value function> ::=
         <current date value function>
       | <current time value function>
       | <current timestamp value function>

<current date value function> ::= CURRENT_DATE

<current time value function> ::=
         CURRENT_TIME [ <left paren> <time precision> <right paren> ]

<current timestamp value function> ::=
         CURRENT_TIMESTAMP [ <left paren> <timestamp precision> <right paren> ]
```

Syntax Rules

1) The data type of a <current date value function> is DATE. The data type of a <current time value function> is TIME WITH TIME ZONE. The data type of a <current timestamp value function> is TIMESTAMP WITH TIME ZONE.

   Note: See the Syntax Rules of Subclause 6.1, "<data type>", for rules governing <time precision> and <timestamp precision>.

2) The null class of a <datetime value function> is the general null class.

Access Rules

None.

General Rules

1) The <datetime value function>s CURRENT_DATE, CURRENT_TIME, and CURRENT_TIMESTAMP respectively return the current date, current time, and current timestamp; the time and timestamp values are returned with time zone displacement equal to the current time zone displacement of the SQL-session.

2) If specified, <time precision> and <timestamp precision> respectively determine the precision of the time or timestamp value returned.

3) If an SQL-statement other than a <compound statement> causes the evaluation of one or more <datetime value function>s, then all such evaluations are effectively performed simultaneously. The time of evaluation of the <datetime value function> during the execution of the SQL-statement is implementation-dependent.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <time precision> or <timestamp precision>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <datetime value function>.

6.11 <reference conversion function>

Function

Specify a function to convert between a reference to, and the value of, an ADT instance.

Format

```
<reference conversion function> ::=
         <ref function>
       | <deref function>

<ref function> ::=
       REF <left paren> { <column reference> | <object reference> } <right paren>

<deref function> ::=
       DEREF <left paren> <value specification> <right paren>
```

Syntax Rules

1) The data type of the <column reference> or <object reference> in a <ref function> shall be an abstract data type that specifies WITH OID.

2) The data type of the <value specification> in a <deref function> shall be object identifier.

Access Rules

None.

General Rules

1) Case:

   a) If REF is specified, then the result is the object identifier of the abstract data type instance identified by the <column reference> or <object reference>.

   b) If DEREF is specified, then the result is the value of the abstract data type instance identified by the object identifier that is the value of the <value specification>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not specify REF or DEREF.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

6.12 <case expression>

Function

Specify a conditional value.

Format

```
<case expression> ::=
        <case abbreviation>
      | <case specification>

<case abbreviation> ::=
        NULLIF <left paren> <value expression> <comma>
               <value expression> <right paren>
      | COALESCE <left paren> <value expression>
               { <comma> <value expression> }... <right paren>

<case specification> ::=
        <simple case>
      | <searched case>

<simple case> ::=
      CASE <case operand>
        <simple when clause>...
        [ <else clause> ]
      END

<searched case> ::=
      CASE
        <searched when clause>...
        [ <else clause> ]
      END

<simple when clause> ::= WHEN <when operand> THEN <result>

<searched when clause> ::= WHEN <search condition> THEN <result>

<else clause> ::= ELSE <result>

<case operand> ::= <value expression>

<when operand> ::= <value expression>

<result> ::= <result expression> | NULL

<result expression> ::= <value expression>
```

Syntax Rules

1) NULLIF (V_1, V_2) is equivalent to the following <case specification>:

   CASE WHEN V_1=V_2 THEN NULL ELSE V_1 END

2) COALESCE (V_1, V_2) is equivalent to the following <case specification>:

   CASE WHEN V_1 IS NOT NULL THEN V_1 ELSE V_2 END

3) COALESCE (V_1, V_2, . . . ,V_n), for n >= 3, is equivalent to the following <case specification>:

   CASE WHEN V_1 IS NOT NULL THEN V_1 ELSE COALESCE (V_2, . . . ,V_n) END

4) If a <case specification> specifies a <simple case>, then let *CO* be the <case operand>:

   a) The data type of each <when operand> *WO* shall be comparable with the data type of the <case operand>.

   b) The <case specification> is equivalent to a <searched case> in which each <searched when clause> specifies a <search condition> of the form ''*CO*=*WO*''.

5) At least one <result> in a <case specification> shall specify a <result expression>.

6) If an <else clause> is not specified, then ELSE NULL is implicit.

7) The data type of a <case specification> is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the data types of all <result expression>s in the <case specification>.

8) The null class of a <case expression> is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the null classes of all result expressions in the <case expression>.

Access Rules

None.

General Rules

1) Case:

   a) If a <result> specifies NULL, then its value is the general null value.

   b) If a <result> specifies a <value expression>, then its value is the value of that <value expression>.

2) Case:

   a) If the <search condition> of some <searched when clause> in a <case specification> is true, then the value of the <case specification> is the value of the <result> of the first (leftmost) <searched when clause> whose <search condition> is true, cast as the data type of the <case specification>.

   b) If no <search condition> in a <case specification> is true, then the value of the <case expression> is the value of the <result> of the explicit or implicit <else clause>, cast as the data type of the <case specification>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any <case expression>.

6.13 <cast specification>

Function

Specify a data conversion.

Format

```
<cast specification> ::=
       CAST <left paren> <cast operand> AS <cast target> <right paren>

<cast operand> ::=
         <value expression>
       | NULL

<cast target> ::=
         <domain name>
       | <data type>
```

Syntax Rules

1) Case:

   a) If a <domain name> is specified, then let *TD* be the <data type> of the specified domain.

   b) If a <data type> is specified, then let *TD* be the specified <data type>.

2) The data type of the result of the <cast specification> is *TD*.

3) If the <cast operand> is a <value expression>, then let *SD* be the underlying data type of the <value expression>.

4) If the <cast operand> is a
   ANSI <value expression> and neither *TD* nor *SD* is a collection type,
   ISO <value expression>,
   then the valid combinations of *TD* and *SD* in a <cast specification> are given by the following table. Y indicates that the combination is syntactically valid without restriction; M indicates that the combination is valid subject to other Syntax Rules in this Subclause being satisfied; and N indicates that the combination is not valid:

SD of <value expression> \ <data type> of *TD*	EN	AN	VC	FC	VB	FB	D	T	TS	YM	DT	ET	BO	ADT	OID
EN	Y	Y	Y	Y	N	N	N	N	N	M	M	Y	N	M	N
AN	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N	M	N
ANSI Only--SQL3															
C	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	M	M
ISO Only---caused by ANSI changes not yet considered by ISO															

C	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	M	N
B	N	N	Y	Y	Y	Y	N	N	N	N	N	N	N	M	N
D	N	N	Y	Y	N	N	Y	N	Y	N	N	N	N	M	N
T	N	N	Y	Y	N	N	N	Y	Y	N	N	N	N	M	N
TS	N	N	Y	Y	N	N	Y	Y	Y	N	N	N	N	M	N
YM	M	N	Y	Y	N	N	N	N	N	Y	N	N	N	M	N
DT	M	N	Y	Y	N	N	N	N	N	N	Y	N	N	M	N
ET	Y	N	Y	Y	N	N	N	N	N	N	N	Y	N	M	N
BO	N	N	Y	Y	N	N	N	N	N	N	N	N	Y	M	N
ADT	M	M	M	M	M	M	M	M	M	M	M	M	M	M	N
ANSI Only--SQL3															
OID	N	N	M	M	N	N	N	N	N	N	N	N	N	N	M
ISO Only---caused by ANSI changes not yet considered by ISO															
OID	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y

Where:

EN = Exact Numeric
AN = Approximate Numeric
C = Character (Fixed- or Variable-length)
FC = Fixed-length Character
VC = Variable-length Character
B = Bit String (Fixed- or Variable-length)
FB = Fixed-length Bit String
VB = Variable-length Bit String
D = Date
T = Time
TS = Timestamp
YM = Year-Month Interval
DT = Day-Time Interval
ET = Enumerated Type
BO = Boolean
ADT = Abstract Data Type
OID = Object Identifier Type

5) If *TD* is an interval and *SD* is exact numeric, then *TD* shall contain only a single <datetime field>.

6) If *TD* is exact numeric and *SD* is an interval, then *SD* shall contain only a single <datetime field>.

7) If *SD* is character string and *TD* is fixed- or variable-length character string, then the character repertoires of *SD* and *TD* shall be the same.

8) If *TD* is a fixed- or variable-length character string, then the collating sequence of the result of the <cast specification> is the default collating sequence for the character repertoire of *TD* and the result of the <cast specification> has the *Coercible* coercibility attribute.

ANSI Only–SQL3

9) If either *SD* or *TD* is an abstract data type, then:

   a) They shall be the same abstract data type; or

   b) There shall be an abstract data type descriptor that identifies a cast function *CP* that has an operand data type *SD* and a return data type *TD*; or

   c) One of *SD* or *TD* shall be a character string type and the other shall be an abstract data type described by an abstract data type descriptor that indicates that the abstract data type is WITH OID VISIBLE.

10) If either *SD* or *TD* is an object identifier type, then:

   a) They shall be the same object identifier type; or

   b) One of *SD* or *TD* shall be a character string type and the other shall be an object identifier type whose associated abstract data type is described by an abstract data type descriptor that indicates that the abstract data type is WITH OID VISIBLE.

11) If *TD* is a collection type, then *SD* shall be a collection type.

12) If *SD* is a collection type, then let *ESD* be the <data type> of the elements of type *SD*.

13) If *TD* is a collection type, then let *ETD* be the <data type> of the elements of type *TD*.

14) If both *SD* and *TD* are collection types, then a CAST of a <value expression> of <data type> *ESD* to <data type> *ETD* shall be permitted by the Syntax Rules of this Subclause.

15) If *SD* is a collection type, then either *TD* shall be the same data type as *ESD*, or both *SD* and *TD* shall be then same collection type, or *SD* shall be a multiset type and *TD* shall be a set type or a list type.

ISO Only—caused by ANSI changes not yet considered by ISO

16) If either *SD* or *TD* is an abstract data type and they are not both the same abstract data type, then there shall be an <abstract data type definition> that contains a <cast clause> whose <operand data type> is *SD* and whose <result data type> is *TD*. Let *CP* be the cast function of that <cast clause>.

17) If either *SD* or *TD* is an object identifier type, then both shall be of object identifier type, and *SD* shall be assignable to *TD*.

Access Rules

1) If <domain name> is specified, then the applicable privileges shall include USAGE.

General Rules

1) If the <cast operand> is a <value expression>, then let *SV* be its value.

2) Case:

   a) If the <cast operand> specifies NULL, then *TV* is the general null value.

   b) If *SV* is a null value, then

      Case:

      i) If a <domain name> is specified that has the same null class as the <cast operand>, then the result is that null value.

      ii) Otherwise the result is the general null value.

   c) Otherwise, let *TV* be the result of the <cast specification> as specified in the remaining General Rules of this Subclause.

3) If either *TD* or *SD* is an abstract data type, then

ISO Only—caused by ANSI changes not yet considered by ISO

both *TD* and *SD* shall have been defined by an <abstract data type definition> that specifies a <cast clause> in which *SD* is the <formal parameter> and *TD* is the <result data type>.

Case:

a) If *TD* and *SD* are the same abstract data type, then let *TV* be *SV*.

ANSI Only–SQL3

b) If a cast function *CP* that casts *SD* to *TD* exists,

ISO Only—caused by ANSI changes not yet considered by ISO

c) If *TD* and *SD* are not the same abstract data type,

then invoke *CP* with a <function invocation>, as appropriate, with <argument> *SV* to obtain a result *TR*.

Case:

i) If *TR* is a null value, then the result of the <cast specification> is the null value.

ii) If *TR* is not a null value, then let *TV* be the non-null value of *TR*.

ANSI Only–SQL3

d) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) If the rules for <literal> in Subclause 5.3, "<literal>", can be applied to *SV* to determine a valid instance of the data type *TD*, then let *TV* be the object identifier of that instance.

ii) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

e) If *TD* is fixed-length character string, then let *LTD* be the length in characters of *TD* and let *Y* be the shortest character string that conforms to the definition of <literal> in Subclause 5.3, "<literal>", such that the interpreted value of *Y* is *SV*.

Case:

i) If *Y* contains any <SQL language character> that is not in the repertoire of $*TD*, then an exception condition is raised: *data exception—invalid character value for cast*.

ii) If the length in characters *LY* of *Y* is equal to *LTD*, then *TV* is *Y*.

iii) If the length in characters *LY* of *Y* is less than *LTD*, then TV) is *Y* extended on the right by $LTD - LY$ <space>s.

iv) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

f) If *TD* is variable-length character string, then let *MLTD* be the maximum length in characters of *TD* and let *Y* be the shortest character string that conforms to the definition of <literal> in Subclause 5.3, "<literal>", such that the interpreted value of *Y* is *SV*.

Case:

i) If *Y* contains any <SQL language character> that is not in the repertoire of *TD*, then an exception condition is raised: *data exception—invalid character value for cast*.

ii) If the length in characters *LY* of *Y* is less than or equal to *MLTD*, then *TV* is *Y*.

iii) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

ANSI Only–SQL3

---

4) If either *TD* or *SD* is an object identifier type, then

Case:

a) If *TD* and *SD* are object identifier types, then let *TV* be *SV*.

b) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) If the rules for <literal> in Subclause 5.3, "<literal>", can be applied to *SV* to determine a valid instance of the data type *TD*, then let *TV* be the object identifier of that instance.

ii) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

c) If *TD* is fixed-length character string, then let *LTD* be the length in characters of *TD* and let *Y* be the shortest character string that conforms to the definition of <literal> in Subclause 5.3, "<literal>", such that the interpreted value of *Y* is *SV*.

Case:

i) If *Y* contains any <SQL language character> that is not in the repertoire of *TD*, then an exception condition is raised: *data exception—invalid character value for cast*.

ii) If the length in characters *LY* of *Y* is equal to *LTD*, then *TV* is *Y*.

iii) If the length in characters *LY* of *Y* is less than *LTD*, then *TV* is *Y* extended on the right by $LTD - LY$ <space>s.

iv) Otherwise, an exception condition is raised: *data exception–string data, right truncation.*

d) If *TD* is variable-length character string, then let *MLTD* be the maximum length in characters of *TD* and let *Y* be the shortest character string that conforms to the definition of <literal> in Subclause 5.3, "<literal>", such that the interpreted value of *Y* is *SV*.

Case:

i) If *Y* contains any <SQL language character> that is not in the repertoire of *TD*, then an exception condition is raised: *data exception—invalid character value for cast.*

ii) If the length in characters *LY* of *Y* is less than or equal to *MLTD*, then *TV* is *Y*.

iii) Otherwise, an exception condition is raised: *data exception—string data, right truncation.*

5) If *SD* is a collection data type, then

Case:

a) If *TD* is the same data type as *ESD*, then

Case:

i) If *SV* is an empty collection, then *TV* is the general null value.

ii) If the cardinality of *SV* is exactly one, then *TV* is the one element of *SV*.

iii) If the cardinality of *SV* is greater than one, then an exception condition is raised: *cardinality violation*.

b) If *SD* is a multiset type and *TD* is a multiset type, then *TV* is a multiset with the same number of elements as *SV*. Each element of *TV* is created by casting the corresponding element of *SV* to the data type *ETD*.

c) If *SD* is a set type and *TD* is a set type, then *TV* is a set with the same number of elements as *SV*. Each element of *TV* is created by casting the corresponding element of *SV* to the data type *ETD*.

d) If *SD* is a list type and *TD* is a list type, then *TV* is a list with same number of elements as *SV*. Each element of *TV* is created by casting the corresponding element of *SV* to the data type *ETD*. The order of elements in *TV* is the same as the order of their corresponding elements in *SV*.

e) If *SD* is a multiset type and *TD* is a set type, then *TV* is a set where the number of elements is equal to the number of elements that would remain in *SV* if redundant duplicate elements were removed. Each element of *TV* is created by casting the corresponding element (after redundant duplicates are removed) of *SV* to the data type *ETD*.

f) If *SD* is a multiset type and *TD* is a list type, then *TV* is a list with the same number of elements as *SV*. Each element of *TV* is created by casting the corresponding element of *SV* to the data type *ETD*. The order of elements in *TV* is implementation-dependent.

ISO Only—caused by ANSI changes not yet considered by ISO

6) If *TD* and *SD* are object identifier types, then let *TV* be *SV*.

7) If *TD* is exact numeric, then

Case:

a) If *SD* is exact numeric or approximate numeric, then

Case:

i) If there is a representation of *SV* in the data type *TD* that does not lose any leading significant digits after rounding or truncating if necessary, then *TV* is that representation. The choice of whether to round or truncate is implementation-defined.

ii) Otherwise, an exception condition is raised: *data exception—numeric value out of range*.

b) If *SD* is character string, then *SV* is replaced by *SV* with any leading or trailing <space>s removed.

Case:

i) If *SV* does not comprise a <signed numeric literal> as defined by the rules for <literal> in Subclause 5.3, "<literal>", then an exception condition is raised: *data exception—invalid character value for cast*.

ii) Otherwise, let *LT* be that <signed numeric literal>. The <cast specification> is equivalent to

CAST (*LT* AS *TD*)

c) If *SD* is an interval data type, then

Case:

i) If there is a representation of *SV* in the data type *TD* that does not lose any leading significant digits, then *TV* is that representation.

ii) Otherwise, an exception condition is raised: *data exception—numeric value out of range*.

d) If *SD* is an enumerated type, then *TV* is the position number of *SV* in *SD*.

8) If *TD* is approximate numeric, then

Case:

a) If *SD* is exact numeric or approximate numeric, then

Case:

i) If there is a representation of *SV* in the data type *TD* that does not lose any leading significant digits after rounding or truncating if necessary, then *TV* is that representation. The choice of whether to round or truncate is implementation-defined.

ii) Otherwise, an exception condition is raised: *data exception—numeric value out of range*.

b) If *SD* is character string, then *SV* is replaced by *SV* with any leading or trailing <space>s removed.

Case:

i) If *SV* does not comprise a <signed numeric literal> as defined by the rules for <literal> in Subclause 5.3, "<literal>", then an exception condition is raised: *data exception—invalid character value for cast*.

ii) Otherwise, let *LT* be that <signed numeric literal>. The <cast specification> is equivalent to

CAST (*LT* AS *TD*)

9) If *TD* is fixed-length character string, then let *LTD* be the length in characters of *TD*.

Case:

a) If *SD* is exact numeric, then let *YP* be the shortest character string that conforms to the definition of <exact numeric literal> in Subclause 5.3, "<literal>", whose scale is the same as the scale of *SD* and whose interpreted value is the absolute value of *SV*.

If *SV* is less than 0, then let *Y* be the result of

'–' || *YP*

Otherwise, let *Y* be *YP*.

Case:

i) If *Y* contains any <SQL language character> that is not in the repertoire of *TD*, then an exception condition is raised: *data exception—invalid character value for cast*.

ii) If the length in characters *LY* of *Y* is equal to *LTD*, then *TV* is *Y*.

iii) If the length in characters *LY* of *Y* is less than *LTD*, then *TV* is *Y* extended on the right by *LTD* − *LY* <space>s.

iv) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

b) If *SD* is approximate numeric, then:

i) Let *YP* be a character string as follows:

Case:

1) If *SV* equals 0, then *YP* is '0E0'.

2) Otherwise, *YP* is the shortest character string that conforms to the definition of <approximate numeric literal> in Subclause 5.3, "<literal>", whose interpreted value is equal to the absolute value of *SV* and whose <mantissa> consists of a single <digit> that is not '0', followed by a <period> and an <unsigned integer>.

ii) If *SV* is less than 0, then let *Y* be the result of

'−' || *YP*

Otherwise, let *Y* be *YP*.

iii) Case:

1) If *Y* contains any <SQL language character> that is not in the repertoire of *TD*, then an exception condition is raised: *data exception—invalid character value for cast*.

2) If the length in characters *LY* of *Y* is equal to *LTD*, then *TV* is *Y*.

3) If the length in characters *LY* of *Y* is less than *LTD*, then *TV* is *Y* extended on the right by *LTD*−*LY* <space>s.

4) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

c) If *SD* is fixed-length character string or variable-length character string, then

Case:

i) If the length in characters of *SV* is equal to *LTD*, then *TV* is *SV*.

ii) If the length in characters of *SV* is larger than *LTD*, then *TV* is the first *LTD* characters of *SV*. If any of the remaining characters of *SV* are non-<space> characters, then a completion condition is raised: *warning—string data, right truncation*.

iii) If the length in characters *M* of *SV* is smaller than *LTD*, then *TV* is *SV* extended on the right by *LTD*−M <space>s.

d) If *SD* is a fixed-length bit string or variable-length bit string, then let *LSV* be the value of BIT_LENGTH(SV) and let *B* be the BIT_LENGTH of the character with the smallest BIT_LENGTH in the form-of-use of *TD*. Let *PAD* be the value of the remainder of the division *LSV*/*B*. Let *NC* be a character whose bits all have the value 0.

If *PAD* is not 0, then append (*B* − *PAD*) 0-valued bits to the least significant end of *SV*; a completion condition is raised: *warning—implicit zero-bit padding*.

Let *SVC* be the possibly padded value of *SV* expressed as a character string without regard to valid character encodings and let *LTDS* be a character string of *LTD* characters of value *NC* characters in the form-of-use of *TD*.

TV is the result of

SUBSTRING (*SVC* || *LTDS* FROM 1 FOR *LTD*)

Case:

i) If the length of *TV* is less than the length of *SVC*, then a completion condition is raised: *warning—string data, right truncation*.

ii) If the length of *TV* is greater than the length of *SVC*, then a completion condition is raised: *warning—implicit zero-bit padding*.

e) If *SD* is a datetime data type or an interval data type, then let *Y* be the shortest character string that conforms to the definition of <literal> in Subclause 5.3, "<literal>", and such that the interpreted value of *Y* is *SV* and the interpreted precision of *Y* is the precision of *SD*.

Case:

i) If *Y* contains any <SQL language character> that is not in the repertoire of *TD*, then an exception condition is raised: *data exception—invalid character value for cast*.

ii) If the length in characters *LY* of *Y* is equal to *LTD*, then *TV* is *Y*.

iii) If the length in characters *LY* of *Y* is less than *LTD*, then *TV* is *Y* extended on the right by *LTD* − *LY* <space>s.

iv) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

f) If *SD* is an enumerated type, then let *Y* be a character string that is the same as the <enumeration name> *SV* in *SD*.

Case:

i) If the length in characters *LY* of *Y* is equal to *LTD*, then *TV* is *Y*.

ii) If the length in characters *LY* of *Y* is less than *LTD*, then *TV* is *Y* extended on the right by *LTD*−*LY* <space>s.

iii) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

g) If *SD* is boolean, then

Case:

i) If *SV* is *true* and *LTD* is not less than 4, then *TV* is 'TRUE' extended on the right by *LTD*−4 <space>s.

ii) If *SV* is *false* and *LTD* is not less than 5, then *TV* is 'FALSE' extended on the right by *LTD*−5 <space>s.

iii) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

10) If *TD* is variable-length character string, then let *MLTD* be the maximum length in characters of *TD*.

Case:

a) If *SD* is exact numeric, then let *YP* be the shortest character string that conforms to the definition of <exact numeric literal> in Subclause 5.3, "<literal>", whose scale is the same as the scale of *SD* and whose interpreted value is the absolute value of *SV*.

If *SV* is less than 0, then let *Y* be the result of

'–' || *YP*

Otherwise, let *Y* be *YP*.

Case:

i) If *Y* contains any <SQL language character> that is not in the repertoire of *TD*, then an exception condition is raised: *data exception—invalid character value for cast*.

ii) If the length in characters *LY* of *Y* is less than or equal to *MLTD*, then *TV* is *Y*.

iii) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

b) If *SD* is approximate numeric, then

i) Let *YP* be a character string as follows:

Case:

1) If *SV* equals 0, then *YP* is '0E0'.

2) Otherwise, *YP* is the shortest character string that conforms to the definition of <approximate numeric literal> in Subclause 5.3, "<literal>", whose interpreted value is equal to the absolute value of *SV* and whose <mantissa> consists of a single <digit> that is not '0', followed by a <period> and an <unsigned integer>.

ii) If *SV* is less than 0, then let *Y* be the result of

'–' || *YP*

otherwise, let *Y* be *YP*.

iii) Case:

1) If *Y* contains any <SQL language character> that is not in the repertoire of *TD*, then an exception condition is raised: *data exception—invalid character value for cast*.

2) If the length in characters *LY* of *Y* is less than or equal to *MLTD*, then *TV* is *Y*.

3) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

c) If *SD* is fixed-length character string or variable-length character string, then

Case:

i) If the length in characters of *SV* is less than or equal to *MLTD*, then *TV* is *SV*.

ii) If the length in characters of *SV* is larger than *MLTD*, then *TV* is the first *MLTD* characters of *SV*. If any of the remaining characters of *SV* are non-<space> characters, then a completion condition is raised: *warning—string data, right truncation*.

d) If *SD* is a fixed-length bit string or variable-length bit string, then let *LSV* be the value of BIT_LENGTH(SV) and let *B* be the BIT_LENGTH of the character with the smallest BIT_LENGTH in the form-of-use of *TD*. Let *PAD* be the value of the remainder of the division *LSV*/*B*.

If *PAD* is not 0, then append (*B* – *PAD*) 0-valued bits to the least significant end of *SV*; a completion condition is raised: *warning—implicit zero-bit padding*.

Let *SVC* be the possible padded value of *SV* expressed as a character string without regard to valid character encodings.

Case:

i) If CHARACTER_LENGTH (*SVC*) is not greater than *MLTD*, then *TV* is *SVC*.

ii) Otherwise, *TV* is the result of

SUBSTRING (*SVC* FROM 1 FOR *MLTD*)

If the length of *TV* is less than the length of *SVC*, then a completion condition is raised: *warning—string data, right truncation*.

e) If *SD* is a datetime data type or an interval data type then let *Y* be the shortest character string that conforms to the definition of <literal> in Subclause 5.3, "<literal>", and such that the interpreted value of *Y* is *SV* and the interpreted precision of *Y* is the precision of *SD*.

Case:

i) If *Y* contains any <SQL language character> that is not in the repertoire of *TD*, then an exception condition is raised: *data exception—invalid character value for cast*.

ii) If the length in characters *LY* of *Y* is less than or equal to *MLTD*, then *TV* is *Y*.

iii) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

f) If *SD* is an enumerated type, then let *Y* be a character string that is the same as the <enumeration name> *SV* in *D*.

Case:

i) If the length in characters *LY* of *Y* is less than or equal to *MLTD*, then *TV* is *Y*.

ii) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

g) If *SD* is boolean, then

Case:

i) If *SV* is *true* and *MLTD* is not less than 4, then *TV* is 'TRUE'.

ii) If *SV* is *false* and *MLTD* is not less than 5, then *TV* is 'FALSE'.

iii) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

11) If *TD* is fixed-length bit string, then let *LTD* be the length in bits of *TD*. Let *BLSV* be the result of BIT_LENGTH(*SV*).

Case:

a) If *BLSV* is equal to *LTD*, then *TV* is *SV* expressed as a bit string with a length in bits of *BLSV*.

b) If *BLSV* is larger than *LTD*, then *TV* is the first *LTD* bits of *SV* expressed as a bit string with a length in bits of *LTD*, and a completion condition is raised: *warning—string data, right truncation*.

c) If *BLSV* is smaller than *LTD*, then *TV* is *SV* expressed as a bit string extended on the right with *LTD* −*BLSV* bits whose values are all 0 and a completion condition is raised: *warning—implicit zero-bit padding*.

12) If *TD* is variable-length bit string, then let *MLTD* be the maximum length in bits of *TD*. Let *BLSV* be the result of BIT_LENGTH(*SV*).

Case:

a) If *BLSV* is less than or equal to *MLTD*, then *TV* is *SV* expressed as a bit string with a length in bits of *BLSV*.

b) If *BLSV* is larger than *MLTD*, then *TV* is the first *MLTD* bits of *SV* expressed as a bit string with a length in bits of *MLTD* and a completion condition is raised: *warning—string data, right truncation*.

13) If *TD* is the datetime data type DATE, then

Case:

a) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) If the rules for <literal> in Subclause 5.3, "<literal>", can be applied to *SV* to determine a valid value of the data type *TD*, then let *TV* be that value.

ii) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

b) If *SD* is a date, then *TV* is *SV*.

c) If *SD* is a timestamp, then *TV* is the year, month, and day <datetime field>s of *SV* adjusted to the implicit or explicit time zone displacement of *SV*.

14) If *TD* is the datetime data type TIME, then

Case:

a) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) If the rules for <literal> in Subclause 5.3, "<literal>", can be applied to *SV* to determine a valid value of the data type *TD*, then let *TV* be that value.

ii) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

b) If *SD* is a time, then *TV* is *SV*. If *TD* is specified WITH TIME ZONE, then *TV* also includes the implicit or explicit time zone displacement of *SV*; otherwise, *TV* is adjusted to the current time zone displacement of the SQL-session.

c) If *SD* is a timestamp, then *TV* is the hour, minute, and second <datetime field>s of *SV*. If *TD* is specified WITH TIME ZONE, then *TV* also includes the implicit or explicit time zone displacement of *SV*; otherwise, *TV* is adjusted to the current time zone displacement of the SQL-session.

15) If *TD* is the datetime data type TIMESTAMP, then

Case:

a) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) If the rules for <literal> in Subclause 5.3, "<literal>", can be applied to *SV* to determine a valid value of the data type *TD*, then let *TV* be that value.

ii) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

b) If *SD* is a date, then the <datetime field>s hour, minute, and second of *TV* are set to 0 and the <datetime field>s year, month, and day of *TV* are set to their respective values in *SV*. If *TD* is specified WITH TIME ZONE, then the time zone fields of *TV* are set to the current time zone displacement of the SQL-session.

c) If *SD* is a time, then the <datetime field>s year, month, and day of *TV* are set to their respective values in an execution of CURRENT_DATE and the <datetime field>s hour, minute, and second of *TV* are set to their respective values in *SV*. If *TD* is specified WITH TIME ZONE, then the time zone fields of *TV* are set to the explicit or implicit time zone interval of *SV*.

d) If *SD* is a timestamp, then *TV* is *SV*.

16) If *TD* is interval, then

Case:

a) If *SD* is exact numeric, then

Case:

i) If the representation of *SV* in the data type *TD* would result in the loss of leading significant digits, then an exception condition is raised: *data exception—interval field overflow*.

ii) Otherwise, *TV* is that representation.

b) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) If the rules for <literal> in Subclause 5.3, "<literal>", can be applied to *SV* to determine a valid value of the data type *TD*, then let *TV* be that value.

ii) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

c) If *SD* is interval and *TD* and *SD* have the same interval precision, then *TV* is *SV*.

d) If *SD* is interval and *TD* and *SD* have different interval precisions, then let *Q* be the least significant <datetime field> of *TD*.

i) Let *Y* be the result of converting *SV* to a scalar in units *Q* according to the natural rules for intervals as defined in the Gregorian calendar.

ii) Normalize *Y* to conform to the datetime qualifier ''*P* TO *Q*'' of *TD*. If this would result in loss of precision of the leading datetime field of *Y*, then an exception condition is raised: *data exception—interval field overflow*.

iii) *TV* is the value of *Y*.

17) If *TD* is an enumerated type, then

Case:

a) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) If *SV* is the same and an <enumeration name> defined within *TD*, then let *TV* be the <enumeration name> corresponding to that character string.

ii) Otherwise, an exception condition is raised: *data exception—invalid enumeration name*.

b) If *SD* is exact numeric, then

Case:

i) If *SV* is negative or *SV* is greater than the largest position number in *TD*, then an exception condition is raised: *data exception—invalid enumeration name*.

ii) Otherwise, *TV* is the <enumeration name> whose position number is *SV* in *TD*.

c) If *SD* is an enumerated type, then *TV* is the value of

CAST (CAST (*SV* AS INTEGER) AS *D*)

18) If *TD* is boolean and *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

a) If the rules for <literal> in Subclause 5.3, "<literal>", can be applied to *SV* to determine a valid value of the data type *TD*, then let *TV* be that value.

b) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

19) If the <cast specification> contains a <domain name> and that <domain name> refers to a domain that contains a <domain constraint> and if *TV* does not satisfy the <check constraint> of the <domain constraint>, then an exception condition is raised: *integrity constraint violation*.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any <cast specification>.

6.14 <value expression>

Function

Specify a value.

Format

```
<value expression> ::=
         <numeric value expression>
       | <string value expression>
       | <datetime value expression>
       | <interval value expression>
       | <enumerated value expression>
       | <boolean value expression>
       | <reference conversion function>
       | <abstract data type value expression>
       | <object identifier value expression>
```

ANSI Only--SQL3

```
       | <table value expression>
       | <collection value expression>
```

```
<abstract data type value expression> ::=
       <value expression primary>

<object identifier value expression> ::=
       <value expression primary>

<collection value expression> ::=
       <value expression primary>

<value expression primary> ::=
         <unsigned value specification>
       | <column reference>
       | <object reference>
       | <set function specification>
```

ANSI Only--SQL3

```
       | <table subquery>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
       | <scalar subquery>
```

```
       | <case expression>
       | <left paren> <value expression> <right paren>
       | <cast specification>
```

Syntax Rules

1) The data type and null class of a <value expression> are the data type and null class of the <numeric value expression>, <string value expression>, <datetime value expression>, <interval value expression>, <enumerated value expression>, <boolean value expression>, <reference conversion function>,
ANSI <collection value expression>,
<abstract data type value expression>,
ANSI <object identifier value expression>, or <table value expression>,
ISO or <object identifier value expression>,
respectively.

2) The data type of <abstract data type value expression> shall be an abstract data type.

3) The data type of <object identifier value expression> shall be an object identifier type.

4) The data type of a <collection value expression> shall be a <collection type>.

5) If the data type of a <value expression primary> is character string, then the collating sequence and coercibility attribute of the <value expression primary> are the collating sequence and coercibility attribute of the <unsigned value specification>, <column reference>, <set function specification>, <scalar subquery>, <case expression>, <value expression>, or <cast specification> immediately contained in the <value expression primary>.

6) Let *C* be some column. Let *VE* be the <value expression>. *C* is an underlying column of *VE* if and only if *C* is identified by some <column reference> contained in *VE*.

Access Rules

None.

General Rules

1) When a <value expression> *V* is evaluated for a row *R* of a table, each reference to a column of that table by a <column reference> *CR* directly contained in *V* is:

   Case:

   a) If *CR* identifies a stored column, then the value of that column in that row.

   b) If *CR* identifies a virtual column, then the result of the SQL function invocation *GF*(*R*), where *GF* is the associated get function.

2) When a <value expression> *V* is evaluated against an object in a table, each reference to an object in that table by an <object reference> directly contained in *V* is a reference to that object.

3) If a <value expression primary> is a <scalar subquery> and the result of the <subquery> is empty, then the result of the <value expression primary> is the general null value.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <value expression> shall not be an <enumerated value expression>.

   b) A <value expression> shall not be a <boolean value expression>.

   c) A <value expression> shall not be a <reference conversion function>.

   d) A <value expression> shall not specify a <collection value expression>.

   e) A <value expression primary> that is a <table subquery> shall satisfy the Syntax Rules and General Rules for a <scalar subquery>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <value expression> shall not be a <datetime value expression>.

   b) A <value expression> shall not be an <interval value expression>.

   c) A <value expression primary> shall not be a <case expression>.

   d) A <value expression primary> shall not be a <cast specification>.

   e) A <value expression primary> shall not be a <scalar subquery> except when the <value expression primary> is simply contained in a <value expression> that is simply contained in the second <row value constructor> of a <comparison predicate>.

6.15 <numeric value expression>

Function

Specify a numeric value.

Format

```
<numeric value expression> ::=
         <term>
       | <numeric value expression> <plus sign> <term>
       | <numeric value expression> <minus sign> <term>

<term> ::=
         <factor>
       | <term> <asterisk> <factor>
       | <term> <solidus> <factor>

<factor> ::=
       [ <sign> ] <numeric primary>

<numeric primary> ::=
         <value expression primary>
       | <numeric value function>
```

Syntax Rules

1) If the data type of both operands of a dyadic arithmetic operator is exact numeric, then the data type of the result is exact numeric, with precision and scale determined as follows:

   a) Let *S1* and *S2* be the scale of the first and second operands respectively.

   b) The precision of the result of addition and subtraction is implementation-defined, and the scale is the maximum of *S1* and *S2*.

   c) The precision of the result of multiplication is implementation-defined, and the scale is *S1* + *S2*.

   d) The precision and scale of the result of division is implementation-defined.

2) If the data type of either operand of a dyadic arithmetic operator is approximate numeric, then the data type of the result is approximate numeric. The precision of the result is implementation-defined.

3) The data type of a <factor> is that of the immediately contained <numeric primary>.

4) The data type of a <numeric primary> shall be numeric.

5) If all the <numeric primary>s that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) If the value of any <numeric primary> simply contained in a <numeric value expression> is a null value, then

   Case:

   a) If the result has the general null class, then the result of the <numeric value expression> is the general null value.

   b) Otherwise, the result of the <numeric value expression> is the minimum of the null values of the operands.

2) If the <numeric value expression> contains only a <numeric primary>, then the value of the <numeric value expression> is the value of the specified <numeric primary>.

3) The monadic arithmetic operators <plus sign> and <minus sign> (+ and −, respectively) specify monadic plus and monadic minus, respectively. Monadic plus does not change its operand. Monadic minus reverses the sign of its operand.

4) The dyadic arithmetic operators <plus sign>, <minus sign>, <asterisk>, and <solidus> (+, −, *, and /, respectively) specify addition, subtraction, multiplication, and division, respectively. If the value of a divisor is zero, then an exception condition is raised: *data exception—division by zero*.

5) If the type of the result of an arithmetic operation is exact numeric, then

   Case:

   a) If the operator is not division and the mathematical result of the operation is not exactly representable with the precision and scale of the result type, then an exception condition is raised: *data exception—numeric value out of range*.

   b) If the operator is division and the approximate mathematical result of the operation represented with the precision and scale of the result type loses one or more leading significant digits after rounding or truncating if necessary, then an exception condition is raised: *data exception—numeric value out of range*. The choice of whether to round or truncate is implementation-defined.

6) If the type of the result of an arithmetic operation is approximate numeric and the exponent of the approximate mathematical result of the operation is not within the implementation-defined exponent range for the result type, then an exception condition is raised: *data exception—numeric value out of range*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

6.16 <string value expression>

Function

Specify a character string value or a bit string value.

Format

```
<string value expression> ::=
         <character value expression>
       | <bit value expression>

<character value expression> ::=
         <concatenation>
       | <character factor>

<concatenation> ::=
       <character value expression> <concatenation operator> <character factor>

<character factor> ::=
       <character primary> [ <collate clause> ]

<character primary> ::=
         <value expression primary>
       | <string value function>

<bit value expression> ::=
         <bit concatenation>
       | <bit factor>

<bit concatenation> ::=
       <bit value expression> <concatenation operator> <bit factor>

<bit factor> ::= <bit primary>

<bit primary> ::=
         <value expression primary>
       | <string value function>
```

Syntax Rules

1) The data type of a <character primary> shall be character string.

2) Character strings of different character repertoires shall not be mixed in a <character value expression>. The character repertoire of a <character value expression> is the character repertoire of its components.

3) Case:

   a) If <concatenation> is specified, then:

      Let *D1* be the data type of the <character value expression> and let *D2* be the data type of the <character factor>. Let *M* be the length in characters of *D1* plus the length in characters of *D2*. Let *VL* be the implementation-defined maximum length of a variable-length character string and let *FL* be the implementation-defined maximum length of a fixed-length character string.

Case:

i) If the data type of the <character value expression> or <character factor> is variable-length character string, then the data type of the <concatenation> is variable-length character string with maximum length equal to the lesser of *M* and *VL*.

ii) If the data type of the <character value expression> and <character factor> is fixed-length character string, then *M* shall not be greater than *FL* and the data type of the <concatenation> is fixed-length character string with length *M*.

b) Otherwise, the data type of the <character value expression> is the data type of the <character factor>.

4) Case:

a) If <character factor> is specified,

Case:

i) If <collate clause> is specified, then the <character value expression> has the collating sequence given in <collate clause>, and has the *Explicit* coercibility attribute.

ii) Otherwise, if <value expression primary> or <string value function> are specified, then the collating sequence and coercibility attribute of the <character factor> are specified in Subclause 6.2, "<value specification> and <target specification>", and Subclause 6.9, "<string value function>", respectively.

b) If <concatenation> is specified, then the collating sequence and the coercibility attribute are determined as specified for dyadic operators in Subclause 4.2.3, "Rules determining collating sequence usage".

5) The data type of a <bit primary> shall be bit string.

6) Case:

a) If <bit concatenation> is specified, then let *D1* be the data type of the <bit value expression>, let *D2* be the data type of the <bit factor>, let *M* be the length in bits of *D1* plus the length in bits of *D2*, let *VL* be the implementation-defined maximum length of a variable-length bit string, and let *FL* be the implementation-defined maximum length of a fixed-length bit string.

Case:

i) If the data type of the <bit value expression> or <bit factor> is variable-length bit string, then the data type of the <bit concatenation> is variable-length bit string with maximum length equal to the lesser of *M* and *VL*.

ii) If the data type of the <bit value expression> and <bit factor> is fixed-length bit string, then *M* shall not be greater than *FL* and the data type of the <bit concatenation> is fixed-length bit string with length *M*.

iii) Otherwise, the data type of a <bit value expression> is the data type of the <bit factor>.

7) If all the <character primary>s or <bit primary>s contained in a <string value expression> that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) If the value of any <character primary> or <bit primary> simply contained in a <string value expression> is a null value, then

   Case:

   a) If the result has the general null class, then the result of the <string value expression> is the general null value.

   b) Otherwise, the result of the <string value expression> is the minimum of the null values of the operands.

2) If <concatenation> is specified, then let *S1* and *S2* be the result of the <character value expression> and <character factor>, respectively.

   Case:

   a) If either *S1* or *S2* is a null value, then the result of the <concatenation> is the general null value.

   b) Otherwise, let *S* be the string consisting of *S1* followed by *S2* and let *M* be the length of *S*.

      Case:

      i) If the data type of either *S1* or *S2* is variable-length character string, then

         Case:

         1) If *M* is less than or equal to *VL*, then the result of the <concatenation> is *S* with length *M*.

         2) If *M* is greater than *VL* and the right-most *M* − *VL* characters of *S* are all the <space> character, then the result of the <concatenation> is the first *VL* characters of *S* with length *VL*.

         3) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

      ii) If the data types of both *S1* and *S2* are fixed-length character string, then the result of the <concatenation> is *S*.

3) If <bit concatenation> is specified, then let *S1* and *S2* be the result of the <bit value expression> and <bit factor>, respectively.

   Case:

   a) If either *S1* or *S2* is a null value, then the result of the <bit concatenation> is the general null value.

b) Otherwise, let *S* be the string consisting of *S1* followed by *S2* and let *M* be the length in bits of *S*.

Case:

i) If the data type of either *S1* or *S2* is variable-length bit string, then

Case:

1) If *M* is less than or equal to *VL*, then the result of the <bit concatenation> is *S* with length *M*.

2) If *M* is greater than *VL* and the right-most *M* − *VL* bits of *S* are all 0-valued, then the result of the <bit concatenation> is the first *VL* bits of *S* with length *VL*.

3) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

ii) If the data types of both *S1* and *S2* are fixed-length bit string, then the result of the <bit concatenation> is *S*.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

b) Conforming Intermediate SQL language shall contain no <bit value expression>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <character value expression> shall not be a <concatenation>.

6.17 <enumerated value expression>

Function

Specify an enumerated type value.

Format

```
<enumerated value expression> ::=
         <domain name> <left paren> <value expression> <right paren>
       | <enumerated primary>

<enumerated primary> ::=
         <value expression primary>
```

Syntax Rules

1) If <domain name> (<value expression>) is specified, then the data type of the <domain name> shall be an enumerated type and the data type of the <value expression> shall be exact numeric with a scale of 0. The data type of the <enumerated value expression> is the enumerated type <domain name>.

2) The data type of an <enumerated primary> shall be an enumerated type. The data type of the result is that enumerated type.

3) The result has the null class of the <enumerated primary>.

Access Rules

None.

General Rules

1) If the value of the <enumerated primary> simply contained in an <enumerated value expression> is a null value, then

   Case:

   a) If the result has the general null class, then the result of the <enumerated value expression> is the general null value.

   b) Otherwise it is that null value.

2) If <domain name>(<value expression>) is specified, then let *SV* be the result of the <value expression>.

   Case:

   a) If *SV* is a null value, then the result of the <enumerated value expression> is the general null value.

   b) If *SV* is negative or if *SV* is greater than the largest position number of <domain name>, then an exception condition is raised: *data exception—invalid enumeration value*.

c) Otherwise, the result of the <enumerated value expression> is the <enumeration literal> of <domain name> whose position number is *SV*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <enumerated value expression>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

6.18 <datetime value expression>

Function

Specify a datetime value.

Format

```
<datetime value expression> ::=
         <datetime term>
       | <interval value expression> <plus sign> <datetime term>
       | <datetime value expression> <plus sign> <interval term>
       | <datetime value expression> <minus sign> <interval term>

<datetime term> ::=
         <datetime factor>

<datetime factor> ::=
         <datetime primary> [ <time zone> ]

<datetime primary> ::=
         <value expression primary>
       | <datetime value function>

<time zone> ::=
       AT <time zone specifier>

<time zone specifier> ::=
         LOCAL
       | TIME ZONE <interval value expression>
```

Syntax Rules

1) The data type of a <datetime primary> shall be datetime.

2) Case:

   a) If the <datetime value expression> is a <datetime term>, then the precision of the result of the <datetime value expression> is the precision of the <datetime value function> or <value expression primary> that it simply contains.

   b) Otherwise, the precision of the result of the <datetime value expression> is the precision of the <datetime value expression> or <datetime term> that it simply contains.

3) If an <interval value expression> or <interval term> is specified, then the <interval value expression> or <interval term> shall only contain <datetime field>s that are contained within the <datetime value expression> or <datetime term>.

4) The data type of the <interval value expression> immediately contained in a <time zone specifier> shall be INTERVAL HOUR TO MINUTE.

5) Case:

   a) If the data type of the <datetime primary> is DATE, then <time zone> shall not be specified.

b) If the data type of the <datetime primary> is TIME or TIMESTAMP and <time zone> is not specified, then ''AT LOCAL'' is implicit.

6) If all the <datetime primary>s contained in a <datetime value expression> that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) If the value of any <datetime primary>, <interval value expression>, <datetime value expression>, or <interval term> simply contained in a <datetime value expression> is a null value, then

Case:

a) If the result has the general null class, then the result of the <datetime value expression> is the general null value.

b) Otherwise, the result of the <datetime value expression> is the minimum of the null values of the operands.

2) If <time zone> is specified or implied and the <interval value expression> immediately contained in <time zone specifier> is null, then the result of the <datetime value expression> is the general null value.

3) If a <datetime value expression> immediately contains the operator + or −, then the result is effectively evaluated as follows:

a) Case:

i) If <datetime value expression> immediately contains the operator + and the <interval value expression> or <interval term> is not negative, or if <datetime value expression> immediately contains the operator − and the <interval term> is negative, then successive <datetime field>s of the <interval value expression> or <interval term> are added to the corresponding fields of the <datetime value expression> or <datetime term>.

ii) Otherwise, successive <datetime field>s of the <interval value expression> or <interval term> are subtracted from the corresponding fields of the <datetime value expression> or <datetime term>.

b) Arithmetic is performed so as to maintain the integrity of the datetime data type that is the result of the <datetime value expression>. This may involve carry from or to the immediately next more significant <datetime field>. If the data type of the <datetime value expression> is TIME, then arithmetic on the HOUR <datetime field> is undertaken modulo 24. If the <interval value expression> or <interval term> is a year-month interval, then the DAY field of the result is the same as the DAY field of the <datetime term> or <datetime value expression>.

c) If, after the preceding step, any <datetime field> of the result is outside the permissible range of values for the field or the result is invalid based on the natural rules for dates and times, then an exception condition is raised: *data exception—datetime field overflow*.

Note: For the permissible range of values for <datetime field>s, see Table 10, "Valid values for fields in datetime items".

4) If <time zone> is specified or implied, then:

a) If LOCAL is specified, then let *TZ* be the current default time zone displacement of the SQL-session. Otherwise, let *TZ* be the value of the <simple value specification> simply contained in the <time zone>.

b) If the value of the <interval value expression> immediately contained in <time zone specifier> is less than INTERVAL −'12:59' or greater than INTERVAL +'13:00', then an exception condition is raised: *data exception—invalid time zone displacement value*.

c) Let *DV* be the value of the <datetime primary> directly contained in the <datetime value expression> expressed as a datetime normalized to UTC.

d) The value of the <datetime value expression> is calculated as:

$$DV - TZ$$

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any <datetime value expression>.

6.19 <interval value expression>

Function

Specify an interval value.

Format

```
<interval value expression> ::=
         <interval term>
       | <interval value expression 1> <plus sign> <interval term 1>
       | <interval value expression 1> <minus sign> <interval term 1>
       | <left paren> <datetime value expression> <minus sign>
             <datetime term> <right paren> <interval qualifier>

<interval term> ::=
         <interval factor>
       | <interval term 2> <asterisk> <factor>
       | <interval term 2> <solidus> <factor>
       | <term> <asterisk> <interval factor>

<interval factor> ::=
       [ <sign> ] <interval primary>

<interval primary> ::=
         <value expression primary> [ <interval qualifier> ]

<interval value expression 1> ::= <interval value expression>

<interval term 1> ::= <interval term>

<interval term 2> ::= <interval term>
```

Syntax Rules

1) The data type of an <interval value expression> is interval. The data type of an <interval primary> shall be interval.

2) An <interval primary> shall specify <interval qualifier> only if the <interval primary> specifies a <dynamic parameter specification>.

3) Case:

   a) If the <interval value expression> simply contains an <interval qualifier>, then the result shall contain the <datetime field>s specified in the <interval qualifier>.

   b) If the <interval value expression> is an <interval term>, then the result of an <interval value expression> contains the same <datetime field>s as the <interval primary>.

   c) If <interval term 1> is specified, then the result contains all the <datetime field>s that are contained within either <interval value expression 1> or <interval term 1>.

4) Case:

   a) If <interval term 1> is a year-month interval, then <interval value expression 1> shall be a year-month interval.

b) If <interval term 1> is a day-time interval, then <interval value expression 1> shall be a day-time interval.

5) If <datetime value expression> is specified, then <datetime value expression> and <datetime term> shall be comparable.

6) If all the <interval primary>s, <datetime value expression>s, <datetime term>s, and <factor>s contained in an <interval value expression> that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) If an <interval term> specifies ''<term> * <interval factor>'', then let *T* and *F* be respectively the value of the <term> and the value of the <interval factor>. The result of the <interval term> is the result of *F* * *T*.

2) If the value of any <interval primary>, <datetime value expression>, <datetime term>, or <factor> that is simply contained in an <interval value expression> is a null value, then

Case:

a) If the result has the general null class, then the result of the <interval value expression> is the general null value.

b) Otherwise, the result of the <interval value expression> is the minimum of the null values of the operands.

3) If the <sign> of an <interval factor> is <minus sign>, then the value of the <interval factor> is the negative of the value of the <interval primary>.

4) If <interval term 2> is specified, then:

a) Let *X* be the value of <interval term 2> and let *Y* be the value of <factor>.

b) Let *P* and *Q* be respectively the most significant and least significant <datetime field>s of <interval term 2>.

c) Let *E* be an exact numeric result of the operation

CAST (CAST (*X* AS INTERVAL *Q*) AS *E1*)

where *E1* is an exact numeric data type of sufficient scale and precision so as to not lose significant digits.

d) Let *OP* be the operator * or / specified in the <interval value expression>.

e) Let *I*, the result of the <interval value expression> expressed in terms of the <datetime field> *Q*, be the result of

CAST ((*E OP Y*) AS INTERVAL *Q*).

f) The result of the <interval value expression> is

CAST (*I* AS INTERVAL *W*)

where *W* is an <interval qualifier> identifying the <datetime field>s *P* TO *Q*, but with <interval leading field precision> such that significant digits are not lost.

5) If <interval term 1> is specified, then let *P* and *Q* be respectively the most significant and least significant <datetime field>s in <interval term 1> and <interval value expression 1>, let *X* be the value of <interval value expression 1>, and let *Y* be the value of <interval term 1>.

a) Let *A* be an exact numeric result of the operation

CAST (CAST (*X* AS INTERVAL *Q*) AS *E1*)

where *E1* is an exact numeric data type of sufficient scale and precision so as to not lose significant digits.

b) Let *B* be an exact numeric result of the operation

CAST (CAST (*Y* AS INTERVAL *Q*) AS *E2*)

where *E2* is an exact numeric data type of sufficient scale and precision so as to not lose significant digits.

c) Let *OP* be the operator + or − specified in the <interval value expression>.

d) Let *I*, the result of the <interval value expression> expressed in terms of the <datetime field> *Q*, be the result of:

CAST ((*A OP B*) AS INTERVAL *Q*)

e) The result of the <interval value expression> is

CAST (*I* AS INTERVAL *W*)

where *W* is an <interval qualifier> identifying the <datetime field>s *P* TO *Q*, but with <interval leading field precision> such that significant digits are not lost.

6) If <datetime value expression> is specified, then let *Y* be the least significant <datetime field> specified by <interval qualifier>. Let *A* be the value represented by <datetime value expression> and let *B* be the value represented by <datetime term>. Evaluation of <interval value expression> proceeds as follows:

a) *A* and *B* are converted to integer scalars *A2* and *B2* respectively in units *Y* as displacements from some implementation-dependent start datetime.

b) The result is determined by effectively computing *A2*−*B2* and then converting the difference to an interval using an <interval qualifier> whose <end field> is *Y* and whose <start field> is sufficiently significant to avoid loss of significant digits. That interval is then converted to an interval using the specified <interval qualifier>, rounding or truncating if necessary. The choice of whether to round or truncate is implementation-defined.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <interval value expression>.

ANSI Only–SQL3

6.20 <table value expression>

Function

Specify a table of values.

****Editor's Note****
It has been pointed out that there is some overlap between Subclause 6.20, "<table value expression>", and Subclause 7.3, "<set value constructor>", and Subclause 7.5, "<list value constructor>". See Possible Problem **280**.

Format

```
<table value expression> ::=
       <table type>
         <left paren>
           [ <value expression> [ { <comma> <value expression> }... ] ]
         <right paren>
```

Syntax Rules

1) Let *TT* be the keyword immediately contained in the specified <table type>.

2) Let *n* be the number of <value expression>s immediately contained in the <table value expression>. If $n > 0$, then let DT_i be the data types and NC_i be the null classes of those <value expression>s in order, $1 <= i <= n$.

3) The data type of a <table value expression> is:

   Case:

   a) If $n = 0$, then SQL_Empty_*TT*.

   b) If $n > 0$, then the data type determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to all the DT_i, $1 <= i <= n$.

4) The null class of a <table value expression> is:

   Case:

   a) If $n = 0$, then the general null class.

   b) If $n > 0$, then the null class determined by applying Subclause 9.3, "Set operation result data types and nullabilities", all the NC_i, $1 <= i <= n$.

Access Rules

None.

General Rules

1) Case:

   a) If $n = 0$, then the result of <table value expression> is an empty table of <table type> *TT*, *i.e.*, an instance of the type SQL_Empty_*TT*.

   b) Otherwise, the result of <table value expression> is a table of <table type> *TT*, *i.e.*, an instance of the type SQL_*TT*(*T*), where *T* is as defined in the Syntax Rules. Let VE_i be the value of each <value expression> immediately contained in the <table value expression>, in order, $1 <= i <= n$, after converting if necessary to the data type *T*.

      Case:

      i) If *TT* is Table, then the result of <table value expression> is a table containing each VE_i.

      ii) If *TT* is Set, then the result of <table value expression> is a set containing the distinct VE_i, *i.e.*, each VE_i, except for elimination of redundant duplicates.

      iii) If *TT* is List, then the result of <table value expression> is a list containing each VE_i in order, $1 <= i <= n$.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <table value expression>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

6.21 <boolean value expression>

Function

Specify a boolean value.

Format

```
<boolean value expression> ::=
         <boolean term>
       | <boolean value expression> OR <boolean term>

<boolean term> ::=
         <boolean factor>
       | <boolean term> AND <boolean factor>

<boolean factor> ::=
       [ NOT ] <boolean primary>

<boolean primary> ::=
         <predicate>
       | <value expression primary>
```

Syntax Rules

1) The data type of a <boolean primary> shall be boolean.

2) Case:

   a) If either operand of a dyadic boolean operator (AND or OR) has the general null class, then the result has null class of the other.

   b) Otherwise,

      Case:

      i) If both operands have the same null class, then the result has that null class.

      ii) Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) The result is derived by the application of the specified boolean operators (''AND'', ''OR'', and ''NOT'') to the results derived from each <boolean primary>. If boolean operators are not specified, then the result of the <boolean value expression> is the result of the specified <boolean primary>.

2) NOT (*true*) is *false*, NOT (*false*) is *true*, and NOT (*null*) is the same null value.

3) Table 12, "Truth table for the AND boolean", and Table 13, "Truth table for the OR boolean", specify the semantics of AND and OR, respectively.

4) Case:

   a) If only one operand is null and the result of the truth table specifies that the result is null, then

      Case:

      i) If the result has the general null class, its value is the general null value.

      ii) Otherwise, its value is the same null value.

   b) Where both operands are null and the result does not have the general null class, then

      Case:

      i) if the operand is AND, then the result is the minimum of such null values.

      ii) Otherwise the result is the maximum of such null values.

Table 12—Truth table for the AND boolean

AND	*true*	*false*	*null*
true	*true*	*false*	*null*
false	*false*	*false*	*false*
null	*null*	*false*	*null*

Table 13—Truth table for the OR boolean

OR	*true*	*false*	*null*
true	*true*	*true*	*true*
false	*true*	*false*	*null*
null	*true*	*null*	*null*

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <boolean value expression>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

7 Query expressions

7.1 <row value constructor>

Function

Specify
ANSI a value or list of
ISO an ordered set of
values to be constructed into a row or partial row.

Format

```
<row value constructor> ::=
         <row value constructor element>
       | <left paren> <row value constructor list> <right paren>
       | <row subquery>

<row value constructor list> ::=
       <row value constructor element>
           [ { <comma> <row value constructor element> }... ]

<row value constructor element> ::=
         <value expression>
       | <null specification>
       | <default specification>

<null specification> ::=
       NULL [ <left paren> <null state> <right paren> ]

<default specification> ::=
       DEFAULT
```

Syntax Rules

1) If a <row value constructor> simply contains a <null specification> or a <default specification>, then the <row value constructor> shall be simply contained in a <query expression> that is simply contained in an <insert statement>.

2) A <row value constructor element> immediately contained in a <row value constructor> shall not be a <value expression> of the form ''<left paren> <value expression> <right paren>''.

   Note: This Rule removes a syntactic ambiguity. A <row value constructor> of this form is permitted, but is parsed in the form ''<left paren> <row value constructor list> <right paren>''.

ANSI Only–SQL3

3) Case:

   a) If the <row value constructor> immediately contains a <row value constructor element> *X*, then the data type of the <row value constructor> is the data type of *X*.

   b) Otherwise, the data types of the column or columns of a <row value constructor> are the data types of the <row value constructor element> or <row value constructor element>s or the columns of the <row subquery> simply contained in the <row value constructor>.

ISO Only—caused by ANSI changes not yet considered by ISO

4) A <row value constructor> that immediately contains a <row value constructor element> *X* is equivalent to a <row value constructor> of the form

   (*X*)

5) The data types of the column or columns of a <row value constructor> are the data types of the <row value constructor element> or <row value constructor element>s or the columns of the <row subquery> simply contained in the <row value constructor>.

6) If a <row value constructor> is derived from a <row subquery>, then the degree of the <row value constructor> is the degree of the table resulting from the <row subquery>; otherwise, the degree of the <row value constructor> is the number of <row value constructor element>s that occur in its specification.

7) If <null state> is specified, then <null state> shall be contained in the null class of the corresponding object column in the containing <insert statement>.

Access Rules

None.

General Rules

1) The value of a <null specification> is a null value.

2) The value of a <default specification> is the default value indicated in the column descriptor for the corresponding column in the explicit or implicit <insert column list> simply contained in the <insert statement>.

3) Case:

ANSI Only–SQL3

   a) If the <row value constructor> immediately contains a <row value constructor element> *X*, then the result of the <row value constructor> is the value of *X*.

b) If a <row value constructor list> is specified, then the result of the <row value constructor> is a row of columns whose *i*-th column has an implementation-dependent name different from the <column name> of all other columns contained in the SQL-statement and whose value is the value of the *i*-th <row value constructor element> in the <row value constructor list>.

c) If the <row value constructor> is a <row subquery>, then:

i) Let *R* be the result of the <row subquery> and let *D* be the degree of *R*.

ii) If the cardinality of *R* is 0, then the result of the <row value constructor> is *D* general null values.

iii) If the cardinality of *R* is 1, then the result of the <row value constructor> is *R*.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <null specification> shall contain no <null state>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) A <row value constructor> that is not simply contained in a <table value constructor> ANSI or an <overlaps predicate> shall not contain more than one <row value constructor element>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <row value constructor element> shall not specify DEFAULT.

7.2 <table value constructor>

Function

Specify a set of <row value constructor>s to be constructed into a table.

Format

```
<table value constructor> ::=
       VALUES <table value constructor list>

<table value constructor list> ::=
       <row value constructor> [ { <comma> <row value constructor> }... ]
```

Syntax Rules

1) All <row value constructor>s shall be of the same degree.

Access Rules

None.

General Rules

1) Let T_i be a table whose *j*-th column has the same data type as the *j*-th <value expression> in the *i*-th <row value constructor> and let T_i contain one row whose *j*-th column has the same value as the *j*-th <value expression> in the *i*-th <row value constructor>.

2) The result of the <table value constructor> is the same as the result of

   T_1 [UNION ALL T_2 [. . . UNION ALL T_n] . . .]

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <table value constructor> shall contain exactly one <row value constructor> that shall be of the form ''(<row value constructor list>)''.

   b) A <table value constructor> shall be the <query expression> of an <insert statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only–SQL3

7.3 <set value constructor>

Function

Specify a collection of elements to be constructed into a multiset.

Format

```
<set value constructor> ::=
       SET <left paren> [ <collection list> ] <right paren>

<collection list> ::= <collection element> [ { <comma> <collection element> }... ]

<collection element> ::= <value expression>
```

Syntax Rules

1) Case:

   a) If <collection list> is omitted, then the <set value constructor> has the data type SET( ).

   b) Otherwise, the data type of the <set value constructor> is SET(*DT*), where *DT* is the data type determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the data types of the <collection element>s immediately contained in the <collection list> of this <set value constructor>.

2) Case:

   a) If no <collection list> is specified, then the <set value constructor> has the general null class.

   b) Otherwise, the null class of the <set value constructor> is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the null classes of the <collection element>s immediately contained in the <collection list> of this <set value constructor>.

Access Rules

None.

General Rules

1) Case:

   a) If <collection list> is omitted, then the result of <set value constructor> is the empty set.

b) Otherwise, the result of <set value constructor> is a set containing the value of each <collection element> immediately contained in the <collection list>, cast as the data type *DT*, with redundant duplicate elements eliminated.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <set value constructor>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only–SQL3

7.4 <multiset value constructor>

Function

Specify a collection of elements to be constructed into a multiset.

Format

```
<multiset value constructor> ::=
       MULTISET <left paren> [ <collection list> ] <right paren>
```

Syntax Rules

1) Case:

   a) If <collection list> is omitted, then the <multiset value constructor> has the data type MULTISET( ).

   b) Otherwise, the data type of the <multiset value constructor> is MULTISET(*DT*), where *DT* is the data type determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the data types of the <collection element>s immediately contained in the <collection list> of this <multiset value constructor>.

2) Case:

   a) If no <collection list> is specified, then the <multiset value constructor> has the general null class.

   b) Otherwise, the null class of the <multiset value constructor> is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the null classes of the <collection element>s immediately contained in the <collection list> of this <multiset value constructor>.

Access Rules

None.

General Rules

1) Case:

   a) If <collection list> is omitted, then the result of <multiset value constructor> is the empty multiset.

b) Otherwise, the result of <multiset value constructor> is a multiset containing the value of each <collection element> immediately contained in the <collection list>, cast as the data type *DT*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <multiset value constructor>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only–SQL3

7.5 <list value constructor>

Function

Specify a collection of elements to be constructed into a list.

Format

```
<list value constructor> ::=
       LIST <left paren> [ <collection list> ] <right paren>
```

Syntax Rules

1) Case:

   a) If <collection list> is omitted, then the <list value constructor> has the data type LIST( ).

   b) Otherwise, the data type of the <list value constructor> is LIST(*DT*), where *DT* is the data type determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the data types of the <collection element>s immediately contained in the <collection list> of this <list value constructor>.

2) Case:

   a) If no <collection list> is specified, then the <list value constructor> has the general null class.

   b) Otherwise, the null class of the <list value constructor> is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the null classes of the <collection element>s immediately contained in the <collection list> of this <list value constructor>.

Access Rules

None.

General Rules

1) Case:

   a) If <collection list> is omitted, then the result of <list value constructor> is the empty list.

   b) Otherwise, the result of <list value constructor> is a list whose *i*-th element is the value of the *i*-th <collection element> immediately contained in the <collection list>, cast as the data type *DT*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <list value constructor>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

7.6 <table expression>

Function

Specify a table or a grouped table.

Format

```
<table expression> ::=
       <from clause>
       [ <where clause> ]
       [ <group by clause> ]
       [ <having clause> ]
```

Syntax Rules

1) The result of a <table expression> is a derived table

ANSI Only–SQL3

---

whose row type is the same as that

ISO Only—caused by ANSI changes not yet considered by ISO

---

in which the descriptor of the *i*-th column is the same as the descriptor of the *i*-th column

---

of the table specified by the <from clause>.

2) Let *C* be some column. Let *TE* be the <table expression>. *C* is an underlying column of *TE* if and only if *C* is an underlying column of some <column reference> contained in *TE*.

Access Rules

None.

General Rules

1) If all optional clauses are omitted, then the result of the <table expression> is the same as the result of the <from clause>. Otherwise, each specified clause is applied to the result of the previously specified clause and the result of the <table expression> is the result of the application of the last specified clause.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If the table identified in the <from clause> is a grouped view, then the <table expression> shall not contain a <where clause>, <group by clause>, or <having clause>.

7.7 <from clause>

Function

Specify a table derived from one or more named tables.

Format

```
<from clause> ::= FROM <table reference> [ { <comma> <table reference> }... ]
```

Syntax Rules

1) Case:

   a) If the <from clause> contains a single <table reference> with no intervening <derived table> or <joined table>, then the descriptor of the result of the <from clause> is the same as the descriptor of the table identified by that <table reference>.

   b) If the <from clause> contains more than one <table reference> with no intervening <derived table> or <joined table>, then the descriptors of the columns of the result of the <from clause> are the descriptors of the columns of the tables identified by the <table reference>s, in the order in which the <table reference>s appear in the <from clause> and in the order in which the columns are defined within each table.

ANSI Only–SQL3

The result of the <from clause> is a multiset table containing objects of a type that is implementation-dependent and different from the type of any other table in the <SQL statement>.

Access Rules

None.

General Rules

1) Case:

   a) If the <from clause> contains a single <table reference> with no intervening <derived table> or <joined table>, then the result of the <from clause> is the table identified by that <table reference>.

   b) If the <from clause> contains more than one <table reference> with no intervening <derived table> or <joined table>, then the result of the <from clause> is the extended Cartesian product of the tables identified by those <table reference>s.

      The extended Cartesian product, *CP*, is the multiset of all rows *R* such that *R* is the concatenation of a row from each of the identified tables in the order in which they are identified. The cardinality of *CP* is the product of the cardinalities of the identified tables. The ordinal position of a column in *CP* is *N* +*S*, where *N* is the ordinal position of that column in the identified table *T* from which it is derived and *S* is the sum of the degrees of the tables

identified before *T* in the <from clause>.

ANSI Only–SQL3

If each of the identified tables has <table type> LIST, then *CP* also has <table type> LIST. The order of the rows *R* is defined as the result of taking each row of the first identified table and concatenating it with the extended Cartesian product of the remaining identified tables in the order in which they were identified.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If the table identified by
   ANSI <table reference>
   ISO <table name>
   is a grouped view, then the <from clause> shall contain exactly one <table reference>.

7.8 <joined table>

Function

Specify a table derived from a Cartesian product, inner or outer join, or union join.

Format

```
<joined table> ::=
         <cross join>
       | <qualified join>
       | <left paren> <joined table> <right paren>

<cross join> ::=
       <table reference> CROSS JOIN <table reference>

<qualified join> ::=
       <table reference> [ NATURAL ] [ <join type> ] JOIN
         <table reference> [ <join specification> ]

<join specification> ::=
         <join condition>
       | <named columns join>
       | <constraint join>

<join condition> ::= ON <search condition>

<named columns join> ::=
       USING <left paren> <join column list> <right paren>

<constraint join> ::=
       | USING PRIMARY KEY
       | USING FOREIGN KEY
       | USING CONSTRAINT <constraint name>

<join type> ::=
         INNER
       | <outer join type> [ OUTER ]
       | UNION

<outer join type> ::=
         LEFT
       | RIGHT
       | FULL

<join column list> ::= <column name list>
```

Syntax Rules

1) Let TR_1 and TR_2 be the first and second <table reference>s of the <joined table>, respectively. Let T_1 and T_2 be the tables identified by TR_1 and TR_2, respectively. Let *TA* and *TB* be the correlation names of TR_1 and TR_2, respectively. Let *CP* be:

   SELECT $*$ FROM TR_1, TR_2

2) If a <qualified join> is specified, then

Case:

a) If NATURAL is specified, then a <join specification> shall not be specified.

b) If UNION is specified, then neither NATURAL nor a <join specification> shall be specified.

c) Otherwise, a <join specification> shall be specified.

3) If a <qualified join> is specified and a <join type> is not specified, then INNER is implicit.

4) If a <qualified join> containing a <join condition> is specified, then;

a) Each <column reference> directly contained in the <search condition> shall unambiguously reference a column of T_1 or T_2 or be an outer reference.

b) If a <value expression> directly contained in the <search condition> is a <set function specification>, then the <joined table> shall be contained in a <having clause> or <select list> and the <set function specification> shall contain a <column reference> that is an outer reference.

Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".
neither NATURAL is specified nor a <join specification> simply containing a <named columns join> is specified, then the descriptors of the columns of the result of the <joined table> are the same as the descriptors of the columns of *CP*.

5) If NATURAL is specified or if a <join specification> simply containing a <named columns join> is specified, then:

a) If NATURAL is specified, then let *common column name* be a <column name> that is the <column name> of exactly one column of T_1 and the <column name> of exactly one column of T_2. T_1 shall not have any duplicate common column names and T_2 shall not have any duplicate common column names. Let *corresponding join columns* refer to all columns of T_1 and T_2 that have common column names, if any.

b) If a <named columns join> is specified, then every <column name> in the <join column list> shall be the <column name> of exactly one column of T_1 and the <column name> of exactly one column of T_2. Let *common column name* be the name of such a column. Let *corresponding join columns* refer to the columns of T_1 and T_2 identified in the <join column list>.

c) Let C_1 and C_2 be a pair of corresponding join columns contained in T_1 and T_2, respectively. C_1 and C_2 shall be comparable.

d) Let *SLCC* be a <select list> of <derived column>s of the form

COALESCE (*TA.C*, *TB.C*) AS *C*

for every column *C* that is a corresponding join column, taken in order of their ordinal positions in T_1.

e) Let SLT_1 be a <select list> of those <column name>s of T_1 that are not corresponding join columns, taken in order of their ordinal positions in T_1, and let SLT_2 be a <select list> of those <column name>s of T_2 that are not corresponding join columns, taken in order of their ordinal positions in T_2.

f) The descriptors of the columns of the result of the <joined table> are the same as the descriptors of the columns of the result of

SELECT *SLCC*, SLT_1, SLT_2 FROM T R_1, TR_2

6) For every column *CR* of the result of the <joined table> that is not a corresponding join column and that corresponds to a column C_1 of T_1, *CR* is *possibly nullable* if any of the following conditions are true:

a) RIGHT, FULL, or UNION is specified, or

b) INNER, LEFT, or CROSS JOIN is specified or implicit and C_1 is possibly nullable.

7) For every column *CR* of the result of the <joined table> that is not a corresponding join column and that corresponds to a column C_2 of T_2, *CR* is *possibly nullable* if any of the following conditions are true:

a) LEFT, FULL, or UNION is specified, or

b) INNER, RIGHT, or CROSS JOIN is specified or implicit and C_2 is possibly nullable.

8) For every column *CR* of the result of the <joined table> that is a corresponding join column and that corresponds to a column C_1 of T_1 and C_2 of T_2, *CR* is *possibly nullable* if any of the following conditions are true:

a) RIGHT, FULL, or UNION is specified and C_1 is possibly nullable, or

b) LEFT, FULL, or UNION is specified and C_2 is possibly nullable.

****Editor's Note****
Paper X3H2-90-023 significantly changed the SQL3 Rules for <joined table>, but did not make any changes to the SQL3 Rules. The SQL3 Rules must be updated correspondingly.

9) Case:

a) If NATURAL is specified, then:

i) Let *common column name* be a <column name> of a column of T_1 which is the same as a <column name> of some column of T_2.

ii) If there are no common column names, then *join columns* are not defined.

iii) If there is at least one common column name, then:

1) Let *join columns* refer to columns of T_1 and T_2 with identical common column names.

2) Let $JC1_i$ and $JC\ 2_i$ be the *i*-th common column name of T_1, taken in the order of their ordinal position in T_1.

b) If PRIMARY KEY is specified, then:

i) The table descriptors for T_1 and T_2 shall each contain a unique constraint descriptor that specifies PRIMARY KEY.

ii) Let *join columns* refer to columns of T_1 and T_2 identified by the <unique column list>s in those <unique specification>s.

iii) Let $JC1_i$ be the name of the *i*-th column in the <unique column list> of T_1. Let $JC2_i$ be the name of the *i*-th column in the <unique column list> of T_2.

c) If FOREIGN KEY is specified, then:

i) There shall be exactly one referential constraint such that either T_1 is the referencing table and T_2 is the referenced table, or T_2 is the referencing table and T_1 is the referenced table.

Note: *referencing table* and *referenced table* are defined in Subclause 11.8, "<referential constraint definition>".

ii) Let *join columns* refer to the referencing columns and referenced columns of that referential constraint.

iii) If T_1 is the referencing table, then:

1) Let $JC1_i$ be the name of the *i*-th referencing column of the referential constraint.

2) Let $JC2_i$ be the name of the *i*-th referenced column of the referential constraint.

iv) If T_1 is the referenced table, then:

1) Let $JC1_i$ be the name of the *i*-th referenced column of the referential constraint.

2) Let $JC2_i$ be the name of the *i*-th referencing column of the referential constraint.

d) If CONSTRAINT is specified, then:

i) Case:

1) If the <table definition> for T_1 contains a <referential constraint definition> identified by <constraint name>, then the <referenced table> in that <referential constraint definition> shall refer to T_2.

2) Otherwise, the <table definition> for T_2 shall contain a <referential constraint definition> identified by <constraint name> whose <referential constraint definition> refers to T_1.

ii) Let *join columns* refer to columns of T_1 and T_2 identified by the <referencing columns> and <referenced table and columns> in that <referential constraint definition>.

iii) If T_1 is the referencing table, then:

1) Let $JC1_i$ be the name of the *i*-th column in the <referencing columns> in the <referential constraint definition>.

2) Let $JC2_i$ be the name of the *i*-th column in the <referenced table and columns> in the <referential constraint definition>.

iv) If T_1 is the referenced table, then:

1) Let $JC1_i$ be the name of the *i*-th column in the <referenced table and columns> in the <referential constraint definition>.

2) Let $JC2_i$ be the name of the *i*-th column in the <referencing columns> in the <referential constraint definition>.

e) Otherwise, *join columns* are not defined.

10) If NATURAL, PRIMARY KEY, FOREIGN KEY, or CONSTRAINT is specified, then

Case:

a) If join columns are defined, then:

i) The data type of the column identified by $JC1_i$ and the data type of the column identified by $JC2_i$ shall be comparable.

ii) Let *D1* be the domain of the column identified by $JC1_i$ and let *D2* be the domain of the column identified by $JC2_i$. If *D1* or *D2* is specified or implied, then both *D1* and *D2* shall specify or imply the same domain.

iii) Let *n* be the number of join columns.

iv) Let *SLCC* be a <select list> of *n* <value expression>s in which the *i*-th <value expression> is:

COALESCE ($T_1.JC1_i$, $T_2.JC2_i$) AS $JC1_i$

v) Let SLT_1 be a <select list> of $D1 - n$ <value expression>s in which the *i*-th <value expression> is the <column name> of the *i*-th column of T_1 that is not included in the join columns, taken in the order of their ordinal position in T_1.

vi) Let SLT_2 be a <select list> of $D2-n$ <value expression>s in which the *i*-th <value expression> is the <column name> of the *i*-th column of T_2 that is not included in the join columns, taken in the order of their ordinal position in T_2.

vii) Let RVT_1 be a <row value constructor> of *n* <value expression>s where the *i*-th <value expression> is $T_1.JC1_i$.

viii) Let RVT_2 be a <row value constructor> of *n* <value expression>s where the *i*-th <value expression> is $T_2.JC2_i$.

ix) Let *JT* be the <join type>.

x) The <joined table> is equivalent to:

(SELECT *SLCC*, SLT_1, SLT_2 FROM T_1 *JT* JOIN T_2 ON RVT_1=RVT_2)

b) Otherwise, the <joined table> is equivalent to:

(SELECT * FROM T_1, T_2)

11) The descriptors of the columns of the result of the <joined table> are the same as the descriptors of the columns of the tables identified by the <table reference>s in the <joined table> in the order in which the <table reference>s appear in the <joined table> and, within the table, in the order in which the columns are defined within that table.

ANSI Only–SQL3

The result of the <joined table> is a multiset table containing objects of a type that is implementation-dependent and different from the type of any other table in the <SQL statement>.

12) Case:

a) If PRIMARY KEY is specified, then:

i) Let *P1* be the <unique column list> of the <unique constraint definition> of T_1 that specifies PRIMARY KEY.

ii) The descriptor of the result of the <joined table> includes the constraint descriptor corresponding to the following <unique constraint definition>:

PRIMARY KEY (*P1*)

b) If FOREIGN KEY or CONSTRAINT is specified and the <table definition> of the referenced table contains a <unique constraint definition> that specifies PRIMARY KEY, then:

i) Let *P1* be the <unique column list> in that <unique constraint definition>.

ii) If *P1* is specified, the the descriptor of the result of the <joined table> includes the constraint descriptor corresponding to the following <unique constraint definition>:

PRIMARY KEY (*P1*)

c) Otherwise, the descriptor of the result of a <joined table> does not include a constraint descriptor corresponding to a <unique constraint definition>.

13) If PRIMARY KEY, FOREIGN KEY, or CONSTRAINT is specified and both T_1 and T_2 are updatable, then the <joined table> is an updatable table. Otherwise, the <joined table> is a read-only table.

Access Rules

None.

General Rules

1) Case:

a) If <join type> is UNION, then let *T* be the empty set.

b) If a <cross join> is specified, then let *T* be the multiset of rows of *CP*.

c) If a <join condition> is specified, then let *T* be the multiset of rows of *CP* for which the specified <search condition> is true.

d) If NATURAL is specified or <named columns join> is specified, then

Case:

i) If there are corresponding join columns, then let *T* be the multiset of rows of *CP* for which the corresponding join columns have equal values.

ii) Otherwise, let *T* be the multiset of rows of *CP*.

2) Let P_1 be the multiset of rows of T_1 for which there exists in *T* some row that is the concatenation of some row R_1 of T_1 and some row R_2 of T_2. Let P_2 be the multiset of rows of T_2 for which there exists in *T* some row that is the concatenation of some row R_1 of T_1 and some row R_2 of T_2.

3) Let U_1 be those rows of T_1 that are not in P_1 and let U_2 be those rows of T_2 that are not in P_2.

4) Let D_1 and D_2 be the degree of T_1 and T_2, respectively. Let X_1 be U_1 extended on the right with D_2 columns containing the general null value. Let X_2 be U_2 extended on the left with D_1 columns containing the general null value.

5) Let $X\ N_1$ and $X\ N_2$ be effective distinct names for X_1 and X_2, respectively. Let *TN* be an effective name for *T*.

   Case:

   a) If INNER or <cross join> is specified, then let *S* be the multiset of rows of *T*.

   b) If LEFT is specified, then let *S* be the multiset of rows resulting from:

      SELECT * FROM *T*
      UNION ALL
      SELECT * FROM X_1

   c) If RIGHT is specified, then let *S* be the multiset of rows resulting from:

      SELECT * FROM *T*
      UNION ALL
      SELECT * FROM X_2

   d) If FULL is specified, then let *S* be the multiset of rows resulting from:

      SELECT * FROM *T*
      UNION ALL
      SELECT * FROM X_1
      UNION ALL
      SELECT * FROM X_2

   e) If UNION is specified, then let *S* be the multiset of rows resulting from:

      SELECT * FROM X_1
      UNION ALL
      SELECT * FROM X_2

6) Let *SN* be an effective name of *S*.

   Case:

   a) If NATURAL is specified or a <named columns join> is specified, then the result of the <joined table> is the multiset of rows resulting from:

      SELECT *SLCC*, SLT_1, SLT_2 FROM *SN*

   b) Otherwise, the result of the <joined table> is *S*.

7) Case:

   a) If PRIMARY KEY is specified, then:

      i) If a row *R* is inserted in <joined table>, then:

         1) Let V_j, V_1, and V_2 be the first *n* values, the next $D_1 - n$ values, and the last $D_2 - n$ values of *R*, respectively.

2) A candidate row CR_1 (CR_2) for T_1 (T_2) is effectively created whose join columns have the value V_j and columns not in join columns have the value V_1 (V_2).

3) If T_1 (T_2) does not contain a row whose value equals the value of CR_1 (CR_2), then CR_1 (CR_2) is inserted into T_1 (T_2).

ii) If a row *R* is deleted from <joined table>, then:

1) Let R_1 (R_2) be the row of T_1 (T_2) from which *R* is derived.

2) If R_1 (R_2) exists, then R_1 (R_2) is deleted from T_1 (T_2).

iii) If a row *R* of *T* is replaced in <joined table> by another row *RR*, then:

1) Let V_j, V_1, and V_2 be the first *n* values, the next D_1-n values, and the last D_2-n values of *RR*, respectively.

2) A candidate row CR_1 (CR_2) for T_1 (T_2) is effectively created whose join columns have the value V_j and columns not in join columns have the value V_1 (V_2).

3) Let R_1 (R_2) be the row of T_1 (T_2) from which *R* is derived.

4) If R_1 (R_2) exists, then R_1 (R_2) is replaced with CR_1 (CR_2).

5) If R_1 (R_2) does not exist and not all of the columns of CR_1 (CR_2) are a null value, then CR_1 (CR_2) is inserted into T_1 (T_2).

b) If FOREIGN KEY or CONSTRAINT is specified, then:

i) If a row *R* is inserted into the <joined table>, then:

1) Let V_j, V_1, and V_2 be the first *n* values, the next D_1-n values, and the last D_2-n values of *R*, respectively.

2) A candidate row CR_1 (CR_2) for T_1 (T_2) is effectively created whose join columns have the value V_j and columns not in join columns have the value V_1 (V_2).

3) If T_1 (T_2) is the referenced table and does not contain a row whose value equals the value of CR_1 (CR_2), then CR_1 (CR_2) is inserted into T_1 (T_2).

4) If T_1 (T_2) is the referencing table and contains a row NMR_1 (NMR_2) that is not one of the matching rows as defined in Subclause 11.8, "<referential constraint definition>", and the columns of T_1 (T_2) that are not in join columns have the value V_1 (V_2), and the <delete rule> of the <referential constraint definition> specifies SET NULL, then NMR_1 (NMR_2) is replaced by CR_1 (CR_2).

5) If T_1 (T_2) is the referencing table and each row in T_1 (T_2) is a matching row, then CR_1 (CR_2) is inserted into T_1 (T_2).

ii) If a row *R* is deleted from *T*, then:

1) Let R_1 (R_2) be the row in T_1 (T_2) from which *R* is derived.

2) If R_1 (R_2) does not exist, then delete R_2 (R_1) from T_2 (T_1).

3) If both R_1 and R_2 exist and T_1 (T_2) is the referencing table, then

Case:

A) If the <referential action> in the <referential constraint definition> specifies SET NULL, then the referencing columns of T_1 (T_2) are set to the general null value.

Note: *referencing columns* are defined in Subclause 11.8, "<referential constraint definition>".

B) Otherwise, R_1 (R_2) is deleted from T_1 (T_2).

iii) If a row *R* of *T* is replaced in <joined table> by another row *RR*, then:

1) Let V_j, V_1, and V_2 be the first *n* values, the next D_1-n values, and the last D_2-n values of *RR*, respectively.

2) A candidate row CR_1 (CR_2) for T_1 (T_2) is effectively created whose join columns have the value V_j and columns not in join columns have the value V_1 (V_2).

3) Let R_1 (R_2) be the row of T_1 (T_2) from which *R* is derived.

4) If R_1 (R_2) does not exist and not all of the columns of CR_1 (CR_2) are a null value, then CR_1 (CR_2) is inserted into T_1 (T_2).

5) If both R_1 and R_2 exist and T_1 (T_2) is the referenced table, then

Case:

A) If the row value of R_1 (R_2) is not equal to the row value of CR_1 (CR_2), then an exception condition is raised: *data exception—invalid update value*.

B) Otherwise, R_1 is replaced by CR_1 and R_2 is replaced by CR_2.

8) If UNION JOIN is specified, then:

a) Let D_1 be the degree of T_1 and let D_2 be the degree of T_2.

b) If a row *R* is inserted, then

Case:

i) If the first D_1 values of *R* are all a null value, and some of the last D_2 values of *R* are not a null value, then *R* is inserted into T_2.

ii) If the last D_2 values of *R* are all a null value, and some of the first D_1 values of *R* are not a null value, then *R* is inserted into T_1.

iii) Otherwise, an exception condition is raised: *data exception—invalid update value*.

c) If a row *R* is deleted from *T*, then

Case:

i) If *R* is derived from a row R_1 of T_1, then delete R_1 from T_1.

ii) If *R* is derived from a row R_2 of T_2, then delete R_2 from T_2.

d) If a row *R* of T_1 is replaced by some row *RR*, then

Case:

i) If *R* is derived from a row R_1 of T_1 and the last D_2 values of *R* are all a null value, then R_1 is replaced by *RR*.

ii) If *R* is derived from a row R_2 of T_2 and the first D_1 values of *R* are all a null value, then R_2 is replaced by *RR*.

iii) Otherwise, an exception condition is raised: *data exception—invalid update value*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <qualified join> shall contain no <constraint join>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <cross join>.

   b) Conforming Intermediate SQL language shall not specify UNION JOIN.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <joined table>.

7.9 <where clause>

Function

Specify a table derived by the application of a <search condition> to the result of the preceding <from clause> or <there is clause>.

Format

```
<where clause> ::= WHERE <search condition>
```

Syntax Rules

ANSI Only–SQL3

1) If the <where clause> is immediately contained in a <table expression>, then let *T* be the result of the preceding <from clause>; otherwise let *T* be the result of the <table reference> immediately contained in the <insert statement> containing the <where clause>. Each <column reference> directly contained in the <search condition> shall unambiguously reference a column of *T* or be an outer reference.

ISO Only—caused by ANSI changes not yet considered by ISO

2) Let *T* be the result of the preceding <from clause>. Each <column reference> directly contained in the <search condition> shall unambiguously reference a column of *T* or be an outer reference.

Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

3) If a <value expression> directly contained in the <search condition> is a <set function specification>, then the <where clause> shall be contained in a <having clause> or <select list> and the <column reference> in the <set function specification> shall be an outer reference.

Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

4) No <column reference> contained in a <subquery> in the <search condition> that references a column of *T* shall be specified in a <set function specification>.

Access Rules

None.

General Rules

1) The <search condition> is applied to each row of *T*. The result of the <where clause> is a table of those rows of *T* for which the result of the <search condition> is true.

ANSI Only–SQL3

If *T* is of <table type> LIST, then the result of the <where clause> is also of <table type> LIST, retaining the order of those rows of *T* that are in the result.

2) Each <subquery> in the <search condition> is effectively executed for each row of *T* and the results used in the application of the <search condition> to the given row of *T*. If any executed <subquery> contains an outer reference to a column of *T*, then the reference is to the value of that column in the given row of *T*.

Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <value expression> directly contained in the <search condition> shall not include a reference to a column that generally contains a <set function specification>.

7.10 <group by clause>

Function

Specify a grouped table derived by the application of the <group by clause> to the result of the previously specified clause.

Format

```
<group by clause> ::=
       GROUP BY <grouping column reference list>

<grouping column reference list> ::=
       <grouping column reference> [ { <comma> <grouping column reference> }... ]

<grouping column reference> ::=
       <column reference> [ <collate clause> ]
```

Syntax Rules

1) If no <where clause> is specified, then let *T* be the result of the preceding <from clause>; otherwise, let *T* be the result of the preceding <where clause>.

2) Each <column reference> in the <group by clause> shall unambiguously reference a column of *T*. A column referenced in a <group by clause> is a grouping column.

3) For every grouping column, if <collate clause> is specified, then the data type of the <column reference> shall be character string. The column descriptor of the corresponding column in the result has the collating sequence specified in <collate clause> and the coercibility attribute *Explicit*.

Access Rules

None.

General Rules

1) The result of the <group by clause> is a partitioning of *T* into a set of groups. The set is the minimum number of groups such that, for each grouping column of each group of more than one row, no two values of that grouping column are distinct.

ANSI Only–SQL3

If *T* is of <table type> LIST, then the result of the <group by clause> is an ordered set of groups, whose order is determined by the order in *T* of the first occurrence of a row from each group. Within each group, the relative order of the rows in *T* is retained.

2) Let *CR* be the <column reference> with <column name> *CN* identifying the grouping column:

Case:

a) If *CR* identifies a stored column, then every row of a given group contains equal values of *CN*. When a <search condition> or <value expression> is applied to a group, *CR* is a reference to the value of *CN*.

b) If *CR* identifies a virtual column, then let *GF* be the associated get function; for any two rows *R1* and *R2* of a given group *GF*(*R1*) = *GF*(*R2*). When a <search condition> or <value expression> is applied to a group, *CR* is a reference to the value returned by the SQL function invocation *GF*(*R*), where *R* is any row in the group.

Note: See the General Rules of Subclause 8.2, "<comparison predicate>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <collate clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

7.11 <having clause>

Function

Specify a grouped table derived by the elimination of groups from the result of the previously specified clause that do not meet the <search condition>.

Format

```
<having clause> ::= HAVING <search condition>
```

Syntax Rules

1) If neither a <where clause> nor a <group by clause> is specified, then let *T* be the result of the preceding <from clause>; if a <where clause> is specified, but a <group by clause> is not specified, then let *T* be the result of the preceding <where clause>; otherwise, let *T* be the result of the preceding <group by clause>. Each <column reference> directly contained in the <search condition> shall unambiguously reference a grouping column of *T* or be an outer reference.

   Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

2) Each <column reference> contained in a <subquery> in the <search condition> that references a column of *T* shall reference a grouping column of *T* or shall be specified within a <set function specification>.

3) The <having clause> is *possibly non-deterministic* if it contains a reference to a column *C* of *T* that has a data type of character string and:

   a) *C* is specified within a <set function specification> that specifies MIN or MAX, or

   b) *C* is a grouping column of *T*.

Access Rules

None.

General Rules

1) Let *T* be the result of the preceding <from clause>, <where clause>, or <group by clause>. If that clause is not a <group by clause>, then *T* consists of a single group and does not have a grouping column.

2) The <search condition> is applied to each group of *T*. The result of the <having clause> is a grouped table of those groups of *T* for which the result of the <search condition> is true.

ANSI Only–SQL3

If the groups of *T* are ordered, then this order is retained in the result of the <having clause>.

3) When the <search condition> is applied to a given group of *T*, that group is the argument or argument source of each <set function specification> directly contained in the <search condition> unless the <column reference> in the <set function specification> is an outer reference.

   Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

4) Each <subquery> in the <search condition> is effectively executed for each group of *T* and the result used in the application of the <search condition> to the given group of *T*. If any executed <subquery> contains an outer reference to a column of *T*, then the reference is to the values of that column in the given group of *T*.

   Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

7.12 <query specification>

Function

Specify a table derived from the result of a <table expression>.

Format

```
<query specification> ::=
       SELECT [ <set quantifier> ] <select list>
       ISO Only---caused by ANSI changes not yet considered by ISO
         [ REFERENCING <referencing values clause> ]
         [ <user-defined updatability clause> ]
         <table expression>

<select list> ::=
         <asterisk>
       | <select sublist> [ { <comma> <select sublist> }... ]

<select sublist> ::=
         <derived column>
       | <qualifier> <period> <asterisk>

<derived column> ::=
       <value expression> [ <as clause> ]

<as clause> ::= [ AS ] <column name>
ISO Only---caused by ANSI changes not yet considered by ISO


<user-defined updatability clause> ::=
       <user-defined insert clause>
       <user-defined update clause>
       <user-defined delete clause>

<user-defined insert clause> ::=
       ON INSERT <user-defined updatability method>

<user-defined delete clause> ::=
       ON DELETE <user-defined updatability method>

<user-defined update clause> ::=
       ON UPDATE <user-defined updatability method>

<user-defined updatability method> ::=
         <function invocation>
       | <delete statement: searched>
       | <update statement: searched>
       | <insert statement>
       | <compound statement>

<referencing values clause> ::=
```

```
    OLD [ AS ] <old values correlation name>
      [ NEW [ AS ] <new values correlation name> ]
  | NEW [ AS ] <new values correlation name>
      [ OLD [ AS ] <old values correlation name> ] } ]
```

Syntax Rules

1) Let *T* be the result of the <table expression>.

2) Let *TQS* be the table that is the result of a <query specification>.

3) The degree of the table specified by a <query specification> is equal to the cardinality of the <select list>.

4) Case:

   a) If the <select list> ‘‘ ∗’’ is simply contained in a <subquery> that is immediately contained in an <exists predicate>, then the <select list> is equivalent to a <value expression> that is an arbitrary <literal>.

   b) Otherwise, the <select list> ‘‘∗’’ is equivalent to a <value expression> sequence in which each <value expression> is a <column reference> that references a column of *T* and each column of *T* (excluding any column with <column name> OID) is referenced exactly once. The columns are referenced in the ascending sequence of their ordinal position within *T*.

5) The <select sublist> ‘‘<qualifier>.∗’’ for some <qualifier> *Q* is equivalent to a <value expression> sequence in which each <value expression> is a <column reference> *CR* that references a column of *T* that is not a common column of a <joined table> (excluding any column with <column name> OID). Each column of *T* that is not a referenced common column shall be referenced exactly once. The columns shall be referenced in the ascending sequence of their ordinal positions within *T*.

   Note: *common column of a <joined table>* is defined in Subclause 7.8, "<joined table>".

6) Let *C* be some column. Let *QS* be the <query specification>. Let DC_i, for *i* ranging from 1 to the number of <derived column>s inclusively, be the *i*-th <derived column> simply contained in the <select list> of *QS*. For all *i*, *C* is an underlying column of DC_i, and of any <column reference> that identifies DC_i, if and only if *C* is an underlying column of the <value expression> of DC_i, or *C* is an underlying column of the <table expression> immediately contained in *QS*.

7) Each <column reference> directly contained in each <value expression> and each <column reference> contained in a <set function specification> directly contained in each <value expression> shall unambiguously reference a column of *T*.

8) If *T* is a grouped table, then let *G* be the set defined by the <grouping column reference list>. Each <column reference> contained in each <value expression> that references a column of *T* shall reference some column *C* such that $G \mapsto C$, or shall be contained in a <set function specification>. If *T* is not a grouped table but some <value expression> contains a <set function specification> that contains a reference to a column of *T* or some <value expression> directly contains a <set function specification> that does not contain an outer reference, then every <column reference> in every <value expression> that references a column of *T* shall be either contained in a <set function specification> or functionally dependent on the empty set.

****Editor's Note****
In discussions of X3H2-93-029/CBR-033, it has been noted that the use of Functional Dependencies in GROUP BY may have a problem when certain schema manipulation operations take place. See Possible Problem **271**.

9) Each column of *TQS* has a column descriptor that includes a data type descriptor that is the same as the data type descriptor of the <value expression> from which the column was derived.

10) Case:

a) If the *i*-th <derived column> in the <select list> specifies an <as clause> that contains a <column name> *CN*, then the <column name> of the *i*-th column of the result is *CN*. *CN* shall not be OID.

b) If the *i*-th <derived column> in the <select list> does not specify an <as clause> and the <value expression> of that <derived column> is a single <column reference>, then the <column name> of the *i*-th column of the result is the <column name> of the column designated by the <column reference>.

c) Otherwise, the <column name> of the *i*-th column of the <query specification> is implementation-dependent and different from the <column name> of any column, other than itself, of a table referenced by any <table reference> contained in the SQL-statement, and shall not be OID.

11) A column of *TQS* is *possibly nullable* if and only if it contains a <column reference> for a column *C* that is possibly nullable, an <indicator parameter>, an <indicator variable>, a <subquery>, CAST NULL AS *X* (*X* represents a <data type> or a <domain name>), SYSTEM_USER, or a <set function specification> that does not contain COUNT.

12) Case:

ANSI Only–SQL3

a) If *QS* is simply contained in a <view definition> whose <table name> is the subject of a <trigger definition> specifying a <trigger action time> INSTEAD OF and a <trigger event> of UPDATE specifying a column *C* of *TQS*, then *C* is updatable.

b) If a column *C* of *TQS* is derived from a <value expression> that comprises a <column reference> *CR*, and if *CR* identifies either a stored column that is updatable or a virtual column with a set function, and no other column of *TQS* is derived from a <value expression> that comprises the same <column reference>, then *C* is updatable.

c) Otherwise, the column of *TQS* is read-only.

• 1 Rule deleted.

13) Let *TREF* be the <table reference>s that are simply contained in the <from clause> of the <table expression>. The *simply underlying table*s of the <query specification> are the tables identified by the <table name>s and <derived table>s contained in *TREF* without an intervening <derived table>.

ISO Only–SQL3

14) A <query specification> *QS* is either *updatable* or *not updatable*. The updatability of *QS* is determined as follows:

a) Let *UT1* denote some underlying table of *QS*, let *CKUT1* be some candidate key of *UT1*, let QSCOLS be the set of columns of *QS*, and let *QSCN1* be some exposed <table name> or exposed <correlation name> whose scope clause is *QS*.

b) If some column *UC* of *UT1* has some *counterpart QC* in *QS*, then let *Q* be the explicit or implicit qualifier of *QC*'s column reference. *QC* is a *counterpart under Q* of *QC*.
Note: *Counterpart* is defined in Subclause 4.18.1, "General rules and definitions".

c) The truth-valued term ''*Q*-to-*U* holds'' is defined as follows:

i) If and only if every member of *CKUT1* has some counterpart under *QSCN1* in *QSCOLS* (so that every row in *QS* corresponds to exactly one row in *UT1*, namely that row in *UT1* that has the same combined value in the columns of *CKUT1* as the row in *QS*), then *Q*-to-*U* holds, with respect to *QS* and *UT1*, under *QSCN1*.

ii) If *Q*-to-*U* holds, with respect to *QS* and *UT1*, under *QSCN1*, then *QS* is said to be *partially updatable* with respect to *UT1* under *QSCN1*.

d) Let *CKQS* be some candidate key of *QS*, let *QSCN2* be some table or correlation name (possibly the same as *QSCN1*) whose scope clsue is *QS*, let *UT2* be some underlying table of *QS*, possibly the same table as *UT1*, and let *UT2COLS* be the set of columns of *UT2*.

e) The truth-valued term ''*U*-to-*Q* holds'' is defined as follows:

i) If and only if every member of *CKQS* is a counterpart under *QSCN2* of some member of *UT2COLS* (so that every row in *UT2* corresponds to exactly one row in *QS*, namely that row in *QS* that has the same combined value in the columns of *CKQS* as the row in *UT2*), then *U*-to-*Q* holds, with respect to *UT2* and *QS*, under *QSCN2*.

ii) If *Q*-to-*U* holds and *U*-to-*Q* holds, with respect to *QS* and *UT2*, under *QSCN2*, then *QS* is said to be *fully updatable* with respect to *UT2* under *QSCN2*.

f) *QS* is updatable if and only if:

i) every column of *QS* has a counterpart, under some qualifier, in some underlying table with respect to which *QS* is partially updatable under that qualifier, and

ii) there is some underlying table with respect to which *QS* is fully updatable under some qualifier.

ANSI Only—caused by ISO changes not yet considered by ANSI

15) A <query specification> *QS* is updatable all of the following conditions hold (otherwise, it is read-only):

a) *QS* does not specify DISTINCT.

b) Every <value expression> contained in the <select list> immediately contained in *QS* consists of a <column reference> identifying an updatable column and no <column reference> appears more than once.

c) The <from clause> immediately contained in the <table expression> immediately contained in *QS* specifies exactly one <table reference> and that <table reference> refers either to a base table or to an updatable derived table.

Note: ''updatable derived table'' is defined in Subclause 6.5, "<table reference>".

d) If the <table expression> immediately contained in *QS* immediately contains a <where clause> *WC*, then no leaf generally underlying table of *QS* shall be a generally underlying table of any <query expression> contained in *WC*.

e) The <table expression> immediately contained in *QS* does not include a <group by clause> or a <having clause>.

---

16) A <query specification> is *possibly non-deterministic* if any of the following conditions are true:

a) The <set quantifier> DISTINCT is specified and one of the columns of *T* has a data type of character string; or

b) The <query specification> directly contains a <having clause> that is possibly non-deterministic; or

c) The <select list> contains a reference to a column *C* of *T* that has a data type of character string and either

i) *C* is specified with a <set function specification> that specifies MIN or MAX, or

ii) *C* is a grouping column of *T*.

ISO Only—caused by ANSI changes not yet considered by ISO

---

17) If a <referencing values clause> is specified, then

a) The <new value correlation name> and the <old value correlation name> shall be different from any other table name or correlation name visible in the scope of the <query specification>.

b) The scope of the <new value correlation name> and the <old value correlation name> is the entire <query specification>.

• 3 Rules deleted.

ANSI Only–SQL3

18) Case:

a) If the <set quantifier> DISTINCT is specified, then *TQS* is a set table.

b) If the <set quantifier> DISTINCT is not specified and *T* is a list table, then *TQS* is a list table.

c) Otherwise, *TQS* is a multiset table.

19) Case:

a) If *TQS* has a single column, then the data type of the objects in *TQS* is the data type of that single column.

b) If *TQS* has more than one column, then the data type of the objects in *TQS* is an implementation-dependent type.

Access Rules

****Editor's Note****
Paper KAW-056/X3H2-92-029 NL#5 said to delete the following rule for SQL2, but this rule is considerably extended for SQL3. Paper KAW-056/X3H2-92-029 NL#5 was not applied. Guidance is requested for what should be done.

1) The applicable privileges shall include SELECT for each <table name> contained in
ANSI the <table expression>
ISO the <table expression> and the <user-defined updatability clause>
as well as SELECT for each <column name> contained in
ANSI the <table expression>.
ISO the <table expression> and the <user-defined updatability clause>.
When a <query specification> is used in a <view definition>, then the <current authorization identifier> shall have privilege descriptors that define the privilege SELECT on all <table name>s contained in the <query expression> as well as SELECT for each <column name> contained in
ANSI the <table expression>.
ISO the <table expression> and the <user-defined updatability clause>.
Such privilege descriptors shall exist, notwithstanding any privilege descriptor granting SELECT privilege to any applicable role of the current user.

Note: The *applicable privileges* for a <table name> are defined in Subclause 10.5, "<privileges>".

General Rules

1) Case:

a) If *T* is not a grouped table, then

Case:

i) If the <select list> contains a <set function specification> that contains a reference to a column of *T* or directly contains a <set function specification> that does not contain an outer reference, then *T* is the argument or argument source of each such <set function

specification> and the result of the <query specification> is a table consisting of 1 row. The *i*-th value of the row is the value specified by the *i*-th <value expression>.

ii) If the <select list> does not include a <set function specification> that contains a reference to *T*, then each <value expression> is applied to each row of *T* yielding a table of *M* rows, where *M* is the cardinality of *T*. The *i*-th column of the table contains the values derived by the evaluation of the *i*-th <value expression>.

ANSI Only–SQL3

If *T* is an ordered table, then the table yielded is an ordered table with its rows in the same order as those of *T* from which they were produced.

Case:

1) If the <set quantifier> DISTINCT is not specified, then the table is the result of the <query specification>.

2) If the <set quantifier> DISTINCT is specified, then the result of the <query specification> is the table derived from that table by the elimination of any redundant duplicate rows.

ANSI Only–SQL3

If *T* is an ordered table, then the order of the remaining rows is retained in the result.

---

b) If *T* is a grouped table, then

Case:

i) If *T* has 0 groups, then the result of the <query specification> is an empty table.

ii) If *T* has one or more groups, then each <value expression> is applied to each group of *T* yielding a table of *M* rows, where *M* is the number of groups in *T*. The *i*-th column of the table contains the values derived by the evaluation of the *i*-th <value expression>. When a <value expression> is applied to a given group of *T*, that group is the argument or argument source of each <set function specification> in the <value expression>.

ANSI Only–SQL3

If *T* is an ordered grouped table, then the table yielded is an ordered table with its rows in the same order as the groups of *T* from which they were produced.

Case:

1) If the <set quantifier> DISTINCT is not specified, then the table is the result of the <query specification>.

2) If the <set quantifier> DISTINCT is specified, then the result of the <query specification> is the table derived from *T* by the elimination of any redundant duplicate rows.

ANSI Only–SQL3

If *T* is an ordered table, the order of the remaining rows is retained in the result.

ISO Only—caused by ANSI changes not yet considered by ISO

2) Let *C* be a column of *TQS*.

3) For every reference to *C* directly contained in the <user-defined updatability clause>,

Case:

a) If the reference to *C* is contained in a <user-defined insert clause>, then the reference to *C* is a reference to the new value being inserted into *C*.

b) If the reference to *C* is contained in a <user-defined delete clause>, then the reference to *C* is a reference to the old value of *C* in the row of *TQS* that is being deleted.

c) If the reference to *C* is contained in a <user-defined update clause>, then

Case:

i) If the reference to *C* is qualified by <new value correlation name> and the row of *TQS* that is being updated has a new value for *C*, then the reference to *C* is a reference to that new value of *C* in the row of *TQS* that is being updated.

ii) Otherwise, the reference to *C* is a reference to the old value of *C* in the row of *TQS* that is being updated.

4) If a <user-defined insert clause is specified, then for each row inserted into *TQS* by an <insert statement>,

Case:

a) If a <function invocation> is specified in the <user-defined insert clause>, then the function identified by <function invocation> is invoked.

b) Otherwise, the <delete statement: searched>, <update statement: searched>, <insert statement>, or <compound statement> specified in the <user-defined insert clause> is executed.

5) If a <user-defined delete clause is specified, then for each row deleted from *TQS* by a <delete statement: positioned> or a <delete statement: searched>,

Case:

a) If a <function invocation> is specified in the <user-defined delete clause>, then the function identified by <function invocation> is invoked.

b) Otherwise, the <delete statement: searched>, <update statement: searched>, <insert statement>, or <compound statement> specified in the <user-defined insert clause> is executed.

6) If a <user-defined update clause> is specified, then for each row updated in *TQS* by an <update statement: positioned> or an <update statement: searched>,

Case:

a) If a <function invocation> is specified in the <user-defined update clause>, then the function identified by <function invocation> is invoked.

b) Otherwise, the <delete statement: searched>, <update statement: searched>, <insert statement>, or <compound statement> specified in the <user-defined insert clause> is executed.

• 4 Rules deleted.

7) If a <user-defined updatability clause> is specified, then *TQS* is effectively a base table with respect to updability rules.

8) Let *UQS* be an updatable <query specification>, let UCN_1, UCN_2, . . . UCN_n be those exposed <table name>s and exposed <correlation name>s whose scope clause is *UQS*, and let UT_1, UT_2, . . . UT_m be those underlying tables of *UQS* with respect to which *UQS* is partially updatable.

If *n*>1, then updating of *UQS* is subject to the constraints that would be implied by WITH CASCADED CHECK OPTION, were *UQS* the <query expression> of a <view definition>.

9) If a row, *UQS_ROW*, is inserted into *UQS*, then:

a) If *UQS_ROW* IN *UQS* is true, then an exception condition is raised: *data exception—row already exists*.

b) If two or more distinct columns in *UQS* have the same counterpart, under the same UCN_i, in UT_j, and *UQS_ROW* does not have the same value in each of those columns, then an exception condition is raised: *data exception—invalid update value*.

c) For each underlying table UT_j, let $UTROW_j$ be the row constructed from the *UQS_ROW* values in columns that have counterparts in UT_j under UCN_i.

Case:

i) If $UTROW_j$ IN UT_j is true, then do nothing;

ii) If there is no row in UT_j that matches $UTROW_j$ in all columns, and *UQS* is not fully updatable with respect to UT_j under UCN_i, then an exception condition is raised: *data exception—invalid update value*.

iii) Otherwise, $UTROW_j$ is expanded as necessary with default values for any missing columns, and an attempt is made to insert the resulting row into UT_j.

10) If a row *UQS_OLD* in *UQS* is updated, then let *UQS_NEW* be the row in UQS that would result from the update.

For every column of *UQS*, if neither *UQS_NEW* has the same value in that column as *UQS_OLD*, nor does that column have a counterpart in some leaf generally underlying table of *UQS* with respect to which *UQS* is fully updatable under some <correlation name> or exposed <table name>, then an exception condition is raised: *data exception—invalid update value*.

For every *j*, *m*>=*j*>=1, if, for some *i*, UCN_i is a qualifier such that *UQS* is fully updatable with respect to UT_j under UCN_i, then:

a) Let $UTOLD_j$ be the row in UT_j that corresponds to *UQS_OLD*, and let $UTNEW_j$ be the row constructed from the *UQS_NEW* values in columns that have counterparts in UT_j under UCN_i, expanded as necessary with values from $UTOLD_j$ for the remaining columns of UT_j.

b) $UTOLD_j$ is replaced by $UTNEW_j$.

11) If a row *UQS_ROW* is deleted from *UQS*, then, for all *j*, m>=j>=1, if, for some *i*, UCN_i is a qualifier such that *UQS* is fully updatable with respect to UT_j under UCN_i, then:

a) Let $UTROW_j$ be the row in UT_j that corresponds to *UQS_ROW*.

b) $UTROW_j$ is deleted from UT_j.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <query specification> shall contain no <user-defined updatability clause> or <referencing new values clause>.

ANSI Only—caused by ISO changes not yet considered by ANSI

b) A <derived column> shall contain no <user-defined column update clause>.

c) A query specification is not updatable if other than a <column reference> or if a <column reference> appears more than once.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) The <set quantifier> DISTINCT shall not be specified more than once in a <query specification>, excluding any <subquery> of that <query specification>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <query specification> is not updatable if the <where clause> of the <table expression> contains a <subquery>.

b) A <select sublist> shall be a <derived column>.

c) If the <table expression> of the <query specification> is a grouped view, then the <select list> shall not contain a <set function specification>.

7.13 <query expression>

Function

Specify a table.

Format

```
<query expression> ::=
```

ANSI Only--SQL3

```
       <possibly updatable query expression> [ <trigger definition>... ]

<possibly updatable query expression> ::=

         <non-join query expression>
       | <joined table>

<non-join query expression> ::=
         <non-join query term>
       | <query expression> UNION  [ ALL ] [ <corresponding spec> ] <query term>
```

ISO Only--SQL3

```
           [ <referencing values clause> ] [ <user-defined updatability clause> ]

       | <query expression> EXCEPT [ ALL ] [ <corresponding spec> ] <query term>
```

ISO Only--SQL3

```
           [ <referencing values clause> ] [ <user-defined updatability clause> ]


<query term> ::=
         <non-join query term>
       | <joined table>

<non-join query term> ::=
         <non-join query primary>
       | <query term> INTERSECT [ ALL ] [ <corresponding spec> ] <query primary>
```

ISO Only--SQL3

```
           [ <referencing values clause> ] [ <user-defined updatability clause> ]

       | <recursive union>

<query primary> ::=
         <non-join query primary>
       | <joined table>
```

```
<non-join query primary> ::=
          <simple table>
        | <left paren> <non-join query expression> <right paren>

<simple table> ::=
          <query specification>
        | <table value constructor>
        | <explicit table>
```

ANSI Only--SQL3

```
        | <collection expression> <explicit table> ::= <table type> <table name>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
<explicit table> ::= TABLE <table name>
```

```
<corresponding spec> ::=
       CORRESPONDING [ BY <left paren> <corresponding column list> <right paren> ]

<corresponding column list> ::= <column name list>
```

ANSI Only--SQL3

```
<collection expression> ::=
       <value expression>
```

Syntax Rules

1) Let *T* be the table specified by the <query expression>.

ANSI Only–SQL3

2) If the <table type> contained in <explicit table> is not TABLE, then it shall be the same as that in the descriptor for the table identified by the <table name> contained in <explicit table>.

3) The <explicit table>

TABLE <table name>

is equivalent to the <query expression>

ANSI Only–SQL3

(SELECT *T* FROM <table name> *T*)

ISO Only—caused by ANSI changes not yet considered by ISO

(SELECT * FROM <table name>)

4) Let *set operator* be UNION [ALL], EXCEPT [ALL], or INTERSECT [ALL].

5) *T* is an updatable table and the <query expression> is updatable if and only if it simply contains a <query expression> *QE* or a <query specification> *QS* and:

ISO Only–SQL3

a) If <user-defined updatability clause> is specified, then *T* is an updatable table.

b) the <query expression> contains *QE* or *QS* without an intervening <non-join query expression> that specified UNION or EXCEPT;

c) the <query expression> contains *QE* or *QS* without an intervening <non-join query term> that specifies INTERSECT; and

d) *QE* or *QS* is updatable.

ANSI Only–SQL3

6) The data type of the <value expression> immediately contained in <collection expression> shall be a <collection type>.

7) Case:

a) If a <simple table> is a <query specification>, then the column descriptor of the *i*-th column of the <simple table> is the same as the column descriptor of the *i*-th column of the <query specification>.

b) If a <simple table> is an <explicit table>, then the column descriptor of the *i*-th column of the <simple table> is the same as the column descriptor of the *i*-th column of the table identified by the <table name> contained in the <explicit table>.

ANSI Only–SQL3

c) If a <simple table> is a <collection expression> and the <data type> of the elements of the <collection expression> is an <abstract data type name>, then the column descriptor of the *i*-th column of the <simple table> is the same as the column descriptor of the *i*-th attribute of the abstract data type identified by that <abstract data type name>.

d) If a <simple table> is a <collection expression> and the <data type> of the elements of the <collection expression> is a <predefined type> or <collection type>, then the <simple table> has one column of that <predefined type> or <collection type> and the name of that column is implementation-dependent and different from the <column name> of any column, other than itself, of a table referenced by any <table reference> contained in the SQL-statement and the <column name> is not OID.

---

e) Otherwise, the column descriptor of the *i*-th column of the <simple table> is the same as the column descriptor of the *i*-th column of the <table value constructor>, except that the <column name> is implementation-dependent and different from the <column name> of any column, other than itself, of a table referenced by any <table reference> contained in the SQL-statement and the <column name> is not OID.

8) Case:

a) If a <non-join query primary> is a <simple table>, then the column descriptor of the *i*-th column of the <non-join query primary> is the same as the column descriptor of the *i*-th column of the <simple table>.

b) Otherwise, the column descriptor of the *i*-th column of the <non-join query primary> is the same as the column descriptor of the *i*-th column of the <non-join query expression>.

9) Case:

a) If a <query primary> is a <non-join query primary>, then the column descriptor of the *i*-th column of the <query primary> is the same as the column descriptor of the *i*-th column of the <non-join query primary>.

b) Otherwise, the column descriptor of the *i*-th column of the <query primary> is the same as the column descriptor of the *i*-th column of the <joined table>.

10) If a set operator is specified in a <non-join query term> or a <non-join query expression>, then let *T1*, *T2*, and *TR* be respectively the first operand, the second operand, and the result of the <non-join query term> or <non-join query expression>. Let *TN1* and *TN2* be the effective names for *T1* and *T2*, respectively.

11) If a set operator is specified in a <non-join query term> or a <non-join query expression>, then let *OP* be the set operator.

Case:

a) If CORRESPONDING is specified, then:

i) Within the columns of *T1*, the same <column name> shall not be specified more than once and within the columns of *T2*, the same <column name> shall not be specified more than once.

ii) At least one column of *T1* shall have a <column name> that is the <column name> of some column of *T2*.

iii) Case:

1) If <corresponding column list> is not specified, then let *SL* be a <select list> of those <column name>s that are <column name>s of both *T1* and *T2* in the order that those <column name>s appear in *T1*.

2) If <corresponding column list> is specified, then let *SL* be a <select list> of those <column name>s explicitly appearing in the <corresponding column list> in the order that these <column name>s appear in the <corresponding column list>. Every <column name> in the <corresponding column list> shall be a <column name> of both *T1* and *T2*.

iv) The <non-join query term> or <non-join query expression> is equivalent to:

(SELECT *SL* FROM *TN1*) *OP* (SELECT *SL* FROM *TN2*)

b) If CORRESPONDING is not specified, then *T1* and *T2* shall be of the same degree.

12) Case:

a) If the <non-join query term> is a <non-join query primary>, then the column descriptor of the *i*-th column of the <non-join query term> is the same as the column descriptor of the *i*-th column of the <non-join query primary>.

b) If the <non-join query term> is a <recursive union>, then the column name of the *i*-th column of the <non-join query term> is that of the *i*-th column of the <recursive union>. The data type of the *i*-th column of the <non-join query term> is the data type of the *i*-th column of the <recursive union>.

c) Otherwise,

i) Case:

1) Let *C* be the <column name> of the *i*-th column of *T1*. If the <column name> of the *i*-th column of *T2* is *C*, then the <column name> of the *i*-th column of *TR* is *C*.

2) Otherwise, the <column name> of the *i*-th column of *TR* is implementation-dependent and different from the <column name> of any column, other than itself, of any table referenced by any <table reference> contained in the SQL-statement.

ii) The data type and null class of the *i*-th column of *TR* is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the data types of the *i*-th column of *T1* and the *i*-th column of *T2*. If the *i*-th column of both *T1* and *T2* are known not nullable, then the *i*-th column of *TR* is known not nullable; otherwise, the *i*-th column of *T* is possibly nullable.

13) Case:

a) If a <query term> is a <non-join query term>, then the column descriptor of the *i*-th column of the <query term> is the same as the column descriptor of the *i*-th column of the <non-join query term>.

b) Otherwise, the column descriptor of the *i*-th column of the <query term> is the same as the column descriptor of the *i*-th column of the <joined table>.

14) Case:

a) If a <non-join query expression> is a <non-join query term>, then the column descriptor of the *i*-th column of the <non-join query expression> is the same as the column descriptor of the *i*-th column of the <non-join query term>.

b) Otherwise,

i) Case:

1) Let *C* be the <column name> of the *i*-th column of *T1*. If the <column name> of the *i*-th column of *T2* is *C*, then the <column name> of the *i*-th column of *TR* is *C*.

2) Otherwise, the <column name> of the *i*-th column of *TR* is implementation-dependent and different from the <column name> of any column, other than itself, of any table referenced by any <table reference> contained in the SQL-statement.

ii) The data type of the *i*-th column of *TR* is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the data types of the *i*-th column of *T1* and the *i*-th column of *T2*. If the *i*-th column of both *T1* and *T2* are known not nullable, then the *i*-th column of *TR* is known not nullable; otherwise, the *i*-th column of *T* is possibly nullable.

15) Case:

a) If a <query expression> is a <non-join query expression>, then the column descriptor of the *i*-th column of the <query expression> is the same as the column descriptor of the *i*-th column of the <non-join query expression>.

b) Otherwise, the column descriptor of the *i*-th column of the <query expression> is the same as the column descriptor of the *i*-th column of the <joined table>.

16) The *simply underlying table*s of a <query expression> are the tables identified by those <table name>s, <query specification>s, and <derived table>s contained in the <query expression> without an intervening
ANSI <derived table>
ISO <derived table>, an intervening <query specification>,
or an intervening <join condition>.

17) A <query expression> is *possibly non-deterministic* if

a) it contains a set operator UNION and ALL is not specified, or if it contains EXCEPT or INTERSECT; and

b) the first or second operand contains a column that has a data type of character string.

18) The *underlying columns* of each column of *QE* and of *QE* itself are defined as follows:

a) A column of a <table value constructor> has no underlying columns.

b) The underlying columns of every *i*-th column of a <simple table> *ST* are the underlying columns of the *i*-th column of the table immediately contained in *ST*.

c) If no set operator is specified, then the underlying columns of every *i*-th column of *QE* are the underlying columns of the *i*-th column of the <simple table> simply contained in *QE*.

d) If a set operator is specified, then the underlying columns of every *i*-th column of *QE* are the underlying columns of the *i*-th column of *T1* and those of the *i*-th column of *T2*.

e) Let *C* be some column. *C* is an underlying column of *QE* if and only if *C* is an underlying column of some column of *QE*.

ISO Only–SQL3

19) If <referencing values clause> is specified, then

a) The <new value correlation name> and the <old value correlation name> shall be different from any other table name or correlation name visible in the scope of the <query expression>.

b) The scope of the <new value correlation name> and the <old value correlation name> is the entire <query expression>.

ANSI Only–SQL3

20) Case:

a) If <query expression> specifies a set operator that specifies ALL, then the result of <query expression> is a multiset table.

b) If <query expression> specifies a set operator and does not specify ALL, then the result of <query expression> is a set table.

c) If no set operator is specified, then the result of <query expression> is the result of the specified <simple table> or <joined table>.

Access Rules

ISO Only–SQL3

1) The applicable privileges shall include SELECT for each <table name> contained in the <user-defined updatability clause> as well as SELECT for each <column name> contained in the <user-defined updatability clause>.

Note: The applicable privileges for a <table name> are defined in Subclause 10.5, "<privileges>".

ANSI Only—caused by ISO changes not yet considered by ANSI

None.

General Rules

ANSI Only–SQL3

1) If a <simple table> is a <collection expression> that has a null value, then the result of <simple table> is an empty table.

2) For each <trigger definition> specified, the General Rules of <trigger definition> are effectively applied to the <derived table> defined by the <query expression>.

3) Case:

   a) If no set operator is specified, then *T* is the result of the specified <simple table> or <joined table>.

   b) If a set operator is specified, then the result of applying the set operator is a table containing the following rows:

      i) Let *R* be a row that is a duplicate of some row in *T1* or of some row in *T2* or both. Let *m* be the number of duplicates of *R* in *T1* and let *n* be the number of duplicates of *R* in *T2*, where $m \geq 0$ and $n \geq 0$.

      ANSI Only–SQL3

      ii) If both *T1* and *T2* are ordered tables, then *T* is an ordered table.

      iii) If ALL is not specified, then

      Case:

      1) If UNION is specified, then

      Case:

      A) If $m > 0$ or $n > 0$, then *T* contains exactly one duplicate of *R*.

      ANSI Only–SQL3

      If *T* is an ordered table, then *R* is derived from its first duplicate occurring in *T1*, or, failing that, from its first duplicate occurring in *T2*.

B) Otherwise, *T* contains no duplicate of *R*.

2) If EXCEPT is specified, then

Case:

A) If $m > 0$ and $n = 0$, then *T* contains exactly one duplicate of *R*.

ANSI Only–SQL3

If *T* is an ordered table, then *R* is derived from its first duplicate occurring in *T1*.

B) Otherwise, *T* contains no duplicate of *R*.

3) If INTERSECT is specified, then

Case:

A) If $m > 0$ and $n > 0$, then *T* contains exactly one duplicate of *R*.

ANSI Only–SQL3

If *T* is an ordered table, then *R* is derived from its first duplicate occurring in *T1*.

B) Otherwise, *T* contains no duplicates of *R*.

iv) If ALL is specified, then

Case:

1) If UNION is specified, then the number of duplicates of *R* that *T* contains is $(m + n)$.

ANSI Only–SQL3

If *T* is an ordered table, then these duplicates of *R* are derived from all its duplicates occurring in *T1* and *T2*.

2) If EXCEPT is specified, then the number of duplicates of *R* that *T* contains is the maximum of $(m - n)$ and 0.

ANSI Only–SQL3

If *T* is an ordered table and contains any duplicates of *R*, they they are derived from the first $m - n$ of its duplicates occurring in *T1*.

3) If INTERSECT is specified, then the number of duplicates of *R* that *T* contains is the minimum of *m* and *n*.

ANSI Only–SQL3

Let *d* be the minimum of *m* and *n*. If *T* is an ordered table and contains any duplicates of *R*, then they are derived from the first *d* of its duplicates occurring in *T1*.

ANSI Only–SQL3

v) If *T* is an ordered table, then its rows are in the order of those of *T1* from which they are derivced, followed by the order of those of *T2* from which they are derived.

Note: See the General Rules of Subclause 8.2, "<comparison predicate>".

4) If a set operator is specified, then for each column whose data type is interval, let *UDT* be in turn the data type of the corresponding column of *T* and let *SV* be the value of the column in each row of the first and second operands. The value of the corresponding column of *T* in the corresponding row of *T* is

CAST (*SV* AS *UDT*)

ISO Only–SQL3

5) For every reference to *C* directly contained in the <user-defined updatability clause>,

Case:

a) If the reference to *C* is contained in a <user-defined insert clause>, then the reference to *C* is a reference to the new value being inserted into *C*.

b) If the reference to *C* is contained in a <user-defined delete clause>, then the reference to *C* is a reference to the old value of *C* in the row of *TQS* that is being deleted.

c) If the reference to *C* is contained in a <user-defined update clause>, then

Case:

i) If the reference to *C* is qualified by <new value correlation name> and the row of *TQS* that is being updated has a new value for *C*, then the reference to *C* is a reference to that new value of *C* in the row of *TQS* that is being updated.

ii) Otherwise, the reference to *C* is a reference to the old value of *C* in the row of *TQS* that is being updated.

6) Case:

ISO Only–SQL3

a) If a <user-defined updatability clause> is specified, then:

i) If a user-defined delete clause> is specified, then for each row deleted from *T* by a <delete statement: positioned> or a <delete statement: searched>,

Case:

1) If a <function invocation> is specified in the <user-defined delete clause>, then the function identified by <function invocation> is invoked.

2) Otherwise, the <delete statement: searched>, <update statement: searched>, <insert statement>, or <compound statement> specified in the <user-defined insert clause> is executed.

ii) If a <user-defined update clause is specified, then for each row updated in *T* by an <update statement: positioned> or an <update statement: searched>,

Case:

1) If a <function invocation> is specified in the <user-defined update clause>, then the function identified by <function invocation> is invoked.

2) Otherwise, the <delete statement: searched>, <update statement: searched>, <insert statement>, or <compound statement> specified in the <user-defined insert clause> is executed.

b) If the set operator UNION is specified, then:

i) If a row *R* is deleted from *T*, then

Case:

1) If *R* is derived from *T1*, then the row from which R is derived is deleted from *T1*.

2) Otherwise, the row from which *R* is derived is deleted from *T2*.

ii) If a row *R* is replaced by some row *RR*, then

Case:

1) If *R* is derived from *T1*, then the row of *T1* from which *R* is derived is replaced by *RR*.

2) Otherwise, the row of *T2* from which *R* is derived is replaced by *RR*.

c) If EXCEPT is specified and a row *R* of *T* is replaced by some row *RR*, then the row of *T1* from which *R* is derived is replaced by *RR*.

d) If INTERSECT is specified, then:

i) If a row *R* is inserted into *T*, then:

1) If *T1* does not contain a row whose value equals the value of *R*, then *R* is inserted into *T1*.

2) If *T1* contains a row whose value equals the value of *R* and no row of *T* is derived from that row, then *R* is inserted into *T1*.

3) If *T2* does not contain a row whose value equals the value of *R*, then *R* is inserted into *T2*.

4) If *T2* contains a row whose value equals the value of *R* and no row of *T* is derived from that row, then *R* is inserted into *T2*.

ii) If a row *R* is replaced by some row *RR*, then:

1) The row of *T1* from which *R* is derived is replaced with *RR*.

2) The row of *T2* from which *R* is derived is replaced with *RR*.

ISO Only–SQL3

7) *T* is effectively a base table with respect to updatability rules.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <non-join query term> shall contain no <recursive union>.

ISO Only–SQL3

b) A <query expression> shall contain no <user-defined updatability clause> or <referencing values clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) A <simple table> shall not be a <table value constructor> except in an <insert statement>.

b) Conforming Intermediate SQL shall contain no <explicit table>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <query expression> shall not specify EXCEPT.

b) A <query term> shall not specify INTERSECT.

c) A <query expression> shall not contain a <joined table>.

d) A <query expression> shall not specify CORRESPONDING.

e) If UNION is specified, then except for column names, the descriptors of the first and second operands shall be identical and the descriptor of the result is identical to the descriptor of the operands.

7.14 <recursive union>

Function

Specify an expression to invoke a limited amount of recursion.

Format

```
<recursive union> ::=
       <left paren> <initial expression>
         RECURSIVE UNION <correlation name list>
             [ <left paren> <recursive column list> <right paren> ]
         <iteration expression>
         [ <search clause> ]
         [ <cycle clause> ]
         [ <limit clause> ] <right paren>

<initial expression> ::= <query expression>

<iteration expression> ::= <query expression>

<correlation name list> ::=
       <correlation name> [ { <comma> <correlation name> }... ]

<recursive column list> ::= <column name list>

<search clause> ::=
       SEARCH <search order> SET <sequence column>

<search order> ::=
         PREORDER
       | { DEPTH | BREADTH } FIRST BY <sort specification list>

<sequence column> ::= <column name>

<cycle clause> ::=
       CYCLE [ <cycle column list> ]
         SET <cycle mark column> [ TO <cycle mark value> ]

<cycle column list> ::=
       <cycle column> [ { <comma> <cycle column> }... ]

<cycle column> ::= <column name>

<cycle mark column> ::= <column name>

<cycle mark value> ::= <value expression>

<limit clause> ::=
       [ RETURN | EXCEPTION ] LIMIT <left paren> <value specification> <right paren>
```

Syntax Rules

1) The scope of each <correlation name> in the <correlation name list> is the <iteration expression>.

2) No <correlation name> shall be specified more than once in the <correlation name list>.

3) The degree of the <query expression> in <initial expression> shall be the same as the degree of the <query expression> in <iteration expression>.

4) The data type of the *i*-th column of the <query expression> in <initial expression> and the data type of the *i*-th column of the <query expression> in <iteration expression> shall be comparable.

5) If a <recursive column list> is specified, then the same <column name> shall not be specified more than once in the <recursive column list>, and the number of <column name>s in the <recursive column list> shall be the same as the degree of the table specified by the <initial expression>.

6) The result description of the <recursive union> includes the column descriptions of the unions of the <initial expression> and the <iteration expression>. If a <recursive column list> is specified, then the name of the *i*-th column of the result is the *i*-th <column name> in the <recursive column list>. If a <recursive column list> is not specified, then the name of the *i*-th column of the result is the name of the *i*-th column of the table specified by the <initial expression>. A *result row* is a row having the result description.

7) If a <search clause> is specified, then:

   a) The <sequence column> shall identify a column of the result description whose data type is INTEGER.

   b) A <column reference> specified in a <sort specification> shall not contain a <qualifier> and shall identify a column of the result description.

   c) An <unsigned integer> specified as a <sort specification> shall not be greater than the degree of the result description.

   Note: <column reference> is the preferred designator for a column in a <sort specification>. The designator <unsigned integer> is a deprecated feature that is supported for compatibility with earlier versions of the
   ANSI American
   ISO International
   Standard. See Annex D, "Deprecated features".

8) If a <cycle clause> is specified, then:

   a) Case:

      i) If a <cycle column list> is specified, then each <column name> in that <cycle column list> shall identify a column of the combination description.

      ii) If a <cycle column list> is not specified, then let *referenced <column name>s* be the <column name>s of the combination description that are referenced by outer references in the <iteration expression>. There shall be at least one referenced <column name>. A <cycle column list> containing exactly one instance of each referenced <column name> is implicit.

      Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

b) The <cycle mark column> shall identify a column of the result whose data type is character string of length 1.

c) If a <cycle mark value> is specified, then its data type shall be character string of length 1. If a <cycle mark value> is not specified, then '1' is implicit.

9) Case:

a) If a <limit clause> is specified, then:

i) The data type of the <value specification> shall be exact numeric with scale 0.

ii) If neither RETURN nor EXCEPTION is specified, then EXCEPTION is implicit.

b) If a <limit clause> is not specified, then EXCEPTION LIMIT (*k*) is implicit, where *k* is an implementation-defined positive integer.

10) Let *N* be the number of <correlation name>s in the <correlation name list>. The combination description of the <recursive union> is a table description that is the concatenation of *N* result descriptions. The *i*-th <correlation name> in the <correlation name list> designates the *i*-th instance of the result description in the combination description. A *combination row* is a row having the combination description.

11) A *linear recursive union* is a <recursive union> whose <correlation name list> contains exactly one <correlation name>. A *non-linear recursive union* is a <recursive union> whose <correlation name list> contains two or more <correlation name>s.

Access Rules

None.

General Rules

1) If a linear recursive union is specified, then:

a) Let *IT* be the set of rows that is the result of evaluating the <initial expression>.

b) If *IT* is the empty set, then that empty set is the result of the <recursive union> and the remaining General Rules are not applicable.

c) Let *L* be the result of the LIMIT <value specification>.

d) If *L* is not positive, then an exception condition is raised: *data exception—invalid limit value*.

e) Let *SEQUENCE* be an INTEGER initialized to 0.

f) Let *RT* be an empty set to contain result rows. Each member of RT will be a result row.

g) Let *S* be an empty push-down stack. Each entry on *S* will be a result row.

h) While TRUE:

i) Depending on whether DEPTH, BREADTH, or PREORDER is specified, the *specified search order* is the order specified by the <sort specification list>, the reverse of the

order specified by the <sort specification list>, or an implementation-defined order, respectively.

ii) Push each row of *IT* onto *S*, in the specified search order.

iii) If *S* is empty, then exit the While loop.

iv) If *SEQUENCE* = *L*, then

Case:

1) If RETURN is specified, then exit the While loop.

2) if EXCEPTION is specified, then an exception condition is raised: *cardinality violation*.

v) Increment *SEQUENCE* by 1.

vi) A row *R1* is the *immediate predecessor* of each row in a set of rows that is the result of evaluating the <iteration expression> with the <correlation name> bound to that row *R1*. A row *R2* is a *predecessor* of a row *R3* if *R1* is the immediate predecessor of *R3* or if *R1* is the immediate predecessor of a row *R2* that is a predecessor of *R3*.

vii) Depending on whether DEPTH, BREADTH, or PREORDER is specified, pop the last, the first, or an arbitrary row *R* of *S*, respectively.

viii) If a <sequence column> is specified, then set the column of *R* identified by the <sequence column> to the value of *SEQUENCE*.

ix) Case:

1) If a <cycle clause> is specified and there is a predecessor of *R* for which each column designated by a <cycle column> is equal to the corresponding column of *R*, then:

A) Set the column of *R* designated by the <cycle mark column> to the value of the <cycle mark value>.

B) Let *IT* be an empty set.

2) Otherwise, let *IT* be the set of rows that is the result of evaluating the <iteration expression> with the <correlation name> bound to *R*.

x) Insert *R* into *RT*.

End While.

i) *RT* is the result of the <recursive union>.

Note: For a linear recursive union, the preceding General Rule and the following General Rule have the same effect. The special Rules for the linear case are, however, simpler than the General Rules that cover both the linear and non-linear cases, and so the special Rules for the linear case are included, even though they are redundant.

2) If a non-linear recursive union is specified, then:

a) Let *IT* be the set of rows that is the result of evaluating the <initial expression>.

b) If *IT* is the empty set, then that empty set is the result of the <recursive union> and the remaining General Rules are not applicable.

c) Let *L* be the result of the LIMIT <value specification>.

d) If *L* is not positive, then an exception condition is raised: *data exception—invalid limit value*.

e) Let *SEQUENCE* be an INTEGER initialized to 0.

f) Let *RT* be an empty set to contain result rows. Each member of RT will be a result row.

g) Let *S* be an empty push-down stack. Each entry on *S* will be either a result row or a combination row.

h) Let *WL* be an empty ordered list. Each entry on *WL* will contain a result row and an *OUTPUT-STATUS* that is either *output* or *not output*. For a result row *RR*, the *entry for RR in WL* refers to the entry of *WL* whose result row is equal to *RR*.

i) Let *R* be an empty combination row.

j) While TRUE:

  i) For result rows, the *natural search order* is the order specified by the <sort specification list>. For combination rows, the (natural search order) is the order specified by repeating the <sort specification list> *N* times, with the *i*-th <correlation name> as a <qualifier> of each <column name> in the *i*-th repetition. Depending on whether DEPTH, BREADTH, or PREORDER is specified, the (specified search order) is the natural search order, the reverse of the natural search order, or an implementation-defined search order, respectively.

  ii) For every row *ITR* of *IT*, in the specified order:

    1) Push *ITR* onto *S* as a result row.

    2) Append *ITR* to *WL* with an *OUTPUT-STATUS* of *not output*.

  iii) If *S* is empty, then exit the While loop.

  iv) Depending on whether DEPTH, BREADTH, or PREORDER is specified, pop the last, the first, or an arbitrary row *R* of *S*, respectively.

  v) Case:

    1) If *R* is a result row, then perform NODE(*R*).

    2) If *R* is a combination row, then perform COMBINATION(*R*).

  End While.

k) *RT* is the result of the <recursive union>.

l) PROCEDURE NODE (*R*):

  Argument *R* is a result row.

  The NODE procedure references *WL* and *S* as global variables.

  i) At the beginning of each iteration of the While of the General Rules of Subclause 7.14, "<recursive union>", each result constituent of combination row *R* is an *immediate constituent* of each result row of *IT*. A result row *R1* is a *predecessor* of a result row *R3*

if *R1* is an immediate predecessor of *R3* or if *R1* is an immediate predecessor of a row *R2* that is a predecessor of *R3*.

ii) A combination row *C1* is a *cycle match* of a combination row *C2* if for each <cycle column> *C*, the value designated by *C* in *C1* is equal to the value designated by *C* in *C2*.

iii) If a <cycle clause> is specified and there is a predecessor of *R* that is a cycle match of *R*, then:

1) Set the column of *R* designated by the <cycle mark column> to the <cycle mark value>.

2) Perform OUTPUT(*R*).

3) Return from the NODE procedure.

iv) Perform OUTPUT(*R*).

v) Let *PT* be the set of all result rows prior to the entry for *R* in *WL*.

vi) Let a *C-term* be an extended Cartesian product with *N* operands, each operand being either *R* or *PT* and at least one operand being *R*. There are $2^N - 1$ different ways to form a *C-term*. Let *CT* be the UNION ALL of the $2^N - 1$ different *C-terms*, with the combination description.

Note: *CT* and the *C-terms* have the following form:

```
          R  x...x R  x R  x R
UNION ALL R  x...x R  x R  x PT
UNION ALL R  x...x R  x PT x R
UNION ALL R  x...x R  x PT x PT
UNION ALL R  x...x PT x R  x R
UNION ALL R  x...x PT x R  x PT
UNION ALL R  x...x PT x PT x R
UNION ALL R  x...x PT x PT x PT
UNION ALL ...
UNION ALL PT x...x PT x PT x R
```

Note: On the first iteration of the While, *PT* is empty and so *CT* is simply the concatenation of *R* with itself *N* times.

Note: If *N* is 1, then there is only one *C-term*, which is *R*, so *CT* is simply *R*.

vii) Each combination row *CR* of *CT* is composed of a sequence of *N* result rows. Each such result row is called a *result constituent* of *CR*.

viii) Push each row of *CT* onto *S* as a combination row, in the specified search order.

ix) Return from the NODE procedure.

m) PROCEDURE COMBINATION (*R*):

Argument *R* is a combination row.

The COMBINATION procedure references *WL* and *SEQUENCE* as global variables.

i) If a <sequence column> is specified, then:

1) Set the <sequence column> of each result constituent of *R* whose entry *AR* on *WL* has *OUTPUT-STATUS* of *output* to the value of the <sequence column> of *AR*.

2) Set the <sequence column> of the *i*-th result constituent of *R* whose entry on *WL* has *OUTPUT-STATUS* of *not output* to *SEQUENCE* + *i*.

ii) Let *IT* be the set of rows that is the result of evaluating the <iteration clause> with the <correlation name>s bound to the result constituents of *R*.

iii) Case:

1) If *IT* is not empty, then for each result constituent *RCC* of *R* whose entry on *WL* has *OUTPUT-STATUS* of *not output*, perform OUTPUT(*RCC*).

2) If *IT* is empty, then perform no action.

3) Return from the COMBINATION procedure.

n) PROCEDURE OUTPUT (*R*):

Argument *R* is a result row.

The OUTPUT procedure references *RT*, *WL*, and *SEQUENCE* as global variables.

i) If the *OUTPUT-STATUS* of *R* on *WL* is *output*, then perform no action and return from the OUTPUT procedure.

ii) If *SEQUENCE* = *L*, then

Case:

1) If RETURN is specified, then exit the <recursive expression> with *RT* as the result.

2) If EXCEPTION is specified, then an exception condition is raised: *cardinality violation*.

iii) Increment *SEQUENCE* by 1.

iv) If a <sequence column> is specified, then set the <sequence column>s of *R* and of the entry for *R* on *WL* to *SEQUENCE*.

v) Insert *R* into *RT*.

vi) Set the *OUTPUT-STATUS* of the entry for *R* on *WL* to *output*.

vii) Return from the OUTPUT procedure.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) Conforming Full SQL language shall contain no <recursive union>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

7.15 <scalar subquery>, <row subquery>, and <table subquery>

Function

Specify a scalar value, a row, or a table derived from a <query expression>.

Format

```
<scalar subquery> ::= <subquery>

<row subquery> ::= <subquery>

<table subquery> ::= <subquery>

<subquery> ::= <left paren> <query expression> <right paren>
```

Syntax Rules

1) The degree of a <scalar subquery> shall be 1.

2) The degree of a <row subquery> shall be greater than 1.

3) The data type and null class of a <scalar subquery> is the data type and null class of the column of the <query expression> immediately contained in the <scalar subquery>.

4) The data types and null classes of the columns of a <row subquery> or <table subquery> are the data types and null classes of the respective columns of the <query expression> immediately contained in the <row subquery> or <table subquery>.

Access Rules

None.

General Rules

1) If the cardinality of a <scalar subquery> or a <row subquery> is greater than 1, then an exception condition is raised: *cardinality violation*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If a <subquery> is contained in a <comparison predicate>, then the <table expression> in the <query specification> shall not contain a <group by clause> or a <having clause> and shall not identify a grouped view.

   b) The <query expression> contained in a <subquery> shall be a <query specification>.

8 Predicates

8.1 <predicate>

Function

Specify a condition that can be evaluated to give a boolean value.

Format

```
<predicate> ::=
        <comparison predicate>
      | <between predicate>
      | <in predicate>
      | <like predicate>
      | <null predicate>
      | <quantified comparison predicate>
      | <exists predicate>
      | <unique predicate>
      | <match predicate>
      | <overlaps predicate>
      | <similar predicate>
      | <quantified predicate>
      | <there is predicate>
      | <distinct predicate>
      | <boolean predicate>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) The result of a <predicate> is a truth value derived according to the General Rules of Subclause 8.2, "<comparison predicate>", Subclause 8.3, "<between predicate>", Subclause 8.4, "<in predicate>", Subclause 8.5, "<like predicate>", Subclause 8.7, "<null predicate>", Subclause 8.8, "<quantified comparison predicate>", Subclause 8.9, "<exists predicate>", Subclause 8.10, "<unique predicate>", Subclause 8.11, "<match predicate>", Subclause 8.6, "<similar predicate>", Subclause 8.13, "<quantified predicate>", Subclause 8.14, "<there is predicate>", or Subclause 8.12, "<overlaps predicate>", as appropriate.

****Editor's Note****

Amelia Carlson has noticed that <quantified comparison predicate> can be reached both directly via <predicate> (see the Format of this Subclause) and indirectly from <quantified predicate> (see the Format of Subclause 8.8, "<quantified comparison predicate>"). It should probably be eliminated from the Format of this Subclause or from the Format of Subclause 8.8, "<quantified comparison predicate>", with appropriate changes to the General Rules and Leveling Rules. See Possible Problem 262.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <similar predicate>.

   b) Conforming Full SQL language shall contain no <quantified predicate>.

   c) Conforming Full SQL language shall contain no <there is predicate>.

   d) Conforming Full SQL language shall contain no <function invocation predicate>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <predicate> shall not be a <match predicate>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <overlaps predicate>.

   b) Conforming Entry SQL language shall not contain any <unique predicate>.

   c) Conforming Entry SQL language shall not contain any <distinct predicate>.

8.2 <comparison predicate>

Function

Specify a comparison of two row values.

Format

```
<comparison predicate> ::=
       <row value constructor> <comp op> <row value constructor>

<comp op> ::=
         <equals operator>
       | <not equals operator>
       | <less than operator>
       | <greater than operator>
       | <less than or equals operator>
       | <greater than or equals operator>
```

Syntax Rules

1) The two <row value constructor>s shall be of the same degree.

2) Let *corresponding values* be values with the same ordinal position in the two <row value constructor>s.

3) The data types of the corresponding values of the two <row value constructor>s shall be comparable.

4) If any pair of respective values has data type object
   ANSI identifier or a <collection type,
   ISO identifier,
   then <comp op> shall be either <equals operator> or <not equals operator>.

5) If any pair of respective values has a data type that is an abstract data type, and <less than operator>, <greater than operator>, <less than or equals operator>, or <greater than or equals operator> is specified, then the abstract data type shall have been defined with an <abstract data type definition> that specifies a <less-than clause> that does not specify NONE.

6) Let *X* be a value in the first <row value constructor> and *Y* be the corresponding value in the second <row value constructor>. If *X* and *Y* have data type character string, then the pair-wise comparison collating sequence used to compare *X* and *Y* is determined by the table for collating sequences for comparisons (Subclause 4.2.3, "Rules determining collating sequence usage"). For any pair of corresponding character strings, let *CS* be the identified collating sequence.

7) If all the values simply contained in the two <row value constructor>s that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) Let *X* and *Y* be any two corresponding <row value constructor element>s. Let *XV* and *YV* be the values represented by *X* and *Y*, respectively.

   Case:

   a) If the data type of *X* is an abstract data type, then equality is determined by invoking the function identified in the <equals clause> of the <abstract data type definition> that defined the abstract data type, and greater-than and less-than relationships are determined by invoking the function identified in the <less-than clause> of the <abstract data type definition> that defined the abstract data type.

      Note: If both ''*X* = *Y*'' and ''*X* < *Y*'' are false, then ''*X* > *Y*'' is true.

   b) If *XV* or *YV* is a null value, then

      Case:

      i) If the result has the general null class, then the result of ''*X* <comp op> *Y*'' is the general null value.

      ii) Otherwise, the result of ''*X* <comp op> *Y*'' is the minimum of the null values of the values simply contained in the two <row value constructor>s.

   c) If *XV* and *YV* are non-null values, then ''*X* <comp op> *Y*'' is true or false as follows:

      i) ''*X* = *Y*'' is true if and only if *XV* and *YV* are equal.

      ii) ''*X* <> *Y*'' is true if and only if *XV* and *YV* are not equal.

      iii) ''*X* < *Y*'' is true if and only if *XV* is less than *YV*.

      iv) ''*X* > *Y*'' is true if and only if *XV* is greater than *YV*.

      v) ''*X* <= *Y*'' is true if and only if *XV* is not greater than *YV*.

      vi) ''*X* >= *Y*'' is true if and only if *XV* is not less than *YV*.

      vii) ''*X* <comp op> *Y*'' is false if and only if ''*X* <comp op> *Y*'' is not true.

2) Numbers are compared with respect to their algebraic value.

3) Enumerated type values are compared with respect to the ordering specified by the <enumerated type>.

4) The comparison of two character strings is determined as follows:

   a) If the length in characters of *X* is not equal to the length in characters of *Y*, then the shorter string is effectively replaced, for the purposes of comparison, with a copy of itself that has been extended to the length of the longer string by concatenation on the right of one or more pad characters, where the pad character is chosen based on *CS*. If *CS* has the NO PAD attribute, then the pad character is an implementation-dependent character different from any character in the character set of *X* and *Y* that collates less than any string under *CS*. Otherwise, the pad character is a <space>.

   b) The result of the comparison of *X* and *Y* is given by the collating sequence *CS*.

c) Depending on the collating sequence, two strings may compare as equal even if they are of different lengths or contain different sequences of characters. When the operations MAX, MIN, DISTINCT, references to a grouping column, and the UNION, EXCEPT, and INTERSECT operators refer to character strings, the specific value selected by these operations from a set of such equal values is implementation-dependent.

Note: If the coercibility attribute of the comparison is *Coercible*, then the collating sequence used is the default defined for the character repertoire. See also other Syntax Rules in this Subclause, Subclause 10.6, "<character set specification>", and Subclause 11.33, "<character set definition>".

5) The comparison of two bit string values, *X* and *Y*, is determined by comparison of their bits with the same ordinal position. If X_i and Y_i are the values of the *i*-th bits of *X* and *Y*, respectively, and if L_X is the length in bits of *X* and L_Y is the length in bits of *Y*, then:

a) *X* is equal to *Y* if and only if $L_X = L_Y$ and $X_i = Y_i$ for all *i*.

b) *X* is less than *Y* if and only if:

i) $L_X < L_Y$ and $X_i = Y_i$ for all *i* less than or equal to L_X; or

ii) $X_i = Y_i$ for all $i < n$ and $X_n = 0$ and $Y_n = 1$ for some *n* less than or equal to the minimum of L_X and L_Y.

6) The comparison of two datetimes is determined according to the interval resulting from their subtraction. Let *X* and *Y* be the two values to be compared and let *H* be the least significant <datetime field> of *X* and *Y*. The result of *X* <comp op> *Y* is defined as:

(*X* − *Y*) *H* <comp op> INTERVAL (0) *H*

Note: Two datetimes are comparable only if they have the same <datetime field>s; see Subclause 4.7.1, "Datetimes".

7) The comparison of two intervals is determined by the comparison of their corresponding values after conversion to integers in some common base unit. Let *X* and *Y* be the two intervals to be compared. Let *A* TO *B* be the specified or implied datetime qualifier of *X* and *C* TO *D* be the specified or implied datetime qualifier of *Y*. Let *T* be the least significant <datetime field> of *B* and *D* and let *U* be a datetime qualifier of the form *T*(*N*), where *N* is an <interval leading field precision> large enough so that significance is not lost in the CAST operation.

X is effectively replaced by CAST (*X* AS INTERVAL *U*).

Y is effectively replaced by CAST (*Y* AS INTERVAL *U*).

The result of the comparison is effectively computed as:

CAST (*X* AS INTEGER) <comp op> CAST (*Y* AS INTEGER)

ANSI Only–SQL3

8) The comparison of two sets or multisets is effectively computed as follows:

a) Let *S1* be the first set or multiset; let *S2* be the second set or multiset.

b) Let *TEMPS1* be a copy of *S1*; let *TEMPS2* be a copy of *S2*.

c) For each element of *TEMPS1*, if that element is equal to an element in *TEMPS2*, then delete that element from *TEMPS1* and *TEMPS2*.

d) Case:

i) If *TEMPS1* is empty and *TEMPS2* is empty, then ''*S1* = *S2*'' is true.

ii) If the number of elements in *TEMPS1* equals the number of elements in *TEMPS2* and the number of non-null elements in *TEMPS1* is less than or equal to the number of null elements in *TEMPS2* and the number of non-null elements in *TEMPS2* is less than or equal to the number of null elements in *TEMPS1*, then ''*S1* = *S2*'' is unknown.

iii) Otherwise, ''*S1* = *S2*'' is false.

9) The comparison of two lists is effectively computed as follows:

a) Let *L1* be the first list; let *L2* be the second list.

b) Let *N1* be the number of elements in *L1*; let *N2* be the number of elements in *L2*.

c) Let *E1i* be the *i*-th element of *L1*; let *E2i* be the *i*-th element of *L2*.

d) If *N1* is not equal to *N2*, then ''*L1* = *L2*'' is false.

e) If *N1* equals zero and *N2* equals zero, then ''*L1* = *L2*'' is true.

f) If ''$E1_i = E2_i$'' is true for all *i* from $i = 1$ to $i = N1$, then ''*L1* = *L2*'' is true.

g) If ''$E1_i = E2_i$'' is false for some *i* from *i=1* to *i=N1*, then ''*L1* = *L2*'' is false.

h) Otherwise, ''*L1* = *L2*'' is unknown.

---

10) Let R_x and R_y be the two <row value constructor>s of the <comparison predicate> and let RX_i and RY_i be the *i*-th <row value constructor element>s of R_x and R_y, respectively. ''R_x <comp op> R_y'' is true, false, or unknown as follows:

a) ''$R_x = R_y$'' is true if and only if $RX_i = RY_i$ for all *i*.

b) ''R_x <> R_y'' is true if and only if RX_i <> RY_i for some *i*.

c) ''$R_x < R_y$'' is true if and only if $RX_i = RY_i$ for all $i < n$ and $RX_n < RY_n$ for some *n*.

d) ''$R_x > R_y$'' is true if and only if $RX_i = RY_i$ for all $i < n$ and $RX_n > RY_n$ for some *n*.

e) ''R_x <= R_y'' is true if and only if $R_x = R_y$ or $R_x < R_y$.

f) ''R_x >= R_y'' is true if and only if $R_x = R_y$ or $R_x > R_y$.

g) ''$R_x = R_y$'' is false if and only if ''R_x <> R_y'' is true.

h) ''R_x <> R_y'' is false if and only if ''$R_x = R_y$'' is true.

i) ''$R_x < R_y$'' is false if and only if ''R_x >= R_y'' is true.

j) ''$R_x > R_y$'' is false if and only if ''R_x <= R_y'' is true.

k) ''R_x <= R_y'' is false if and only if ''$R_x > R_y$'' is true.

l) ''R_x >= R_y'' is false if and only if ''$R_x < R_y$'' is true.

m) ''R_x <comp op> R_y'' is unknown if and only if ''R_x <comp op> R_y'' is neither true nor false.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.3 <between predicate>

Function

Specify a range comparison.

Format

```
<between predicate> ::=
       <row value constructor> [ NOT ] BETWEEN
         <row value constructor> AND <row value constructor>
```

Syntax Rules

1) The three <row value constructor>s shall be of the same degree.

2) Let respective values be values with the same ordinal position in the three <row value constructor>s.

3) The data types of the respective values of the three <row value constructor>s shall be comparable.

4) Let *X*, *Y*, and *Z* be the first, second, and third <row value constructor>s, respectively.

5) ''*X* NOT BETWEEN *Y* AND *Z*'' is equivalent to ''NOT (*X* BETWEEN *Y* AND *Z*)''.

6) ''*X* BETWEEN *Y* AND *Z*'' is equivalent to ''*X*>=*Y* AND *X*<=*Z*''.

7) If all the values simply contained in the two <row value constructor>s that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

8.4 <in predicate>

Function

Specify a quantified comparison.

Format

```
<in predicate> ::=
       <row value constructor>
         [ NOT ] IN <in predicate value>

<in predicate value> ::=
         <table subquery>
       | <left paren> <in value list> <right paren>

<in value list> ::=
       <value expression> { <comma> <value expression> }...
```

Syntax Rules

1) Let *IVL* be an <in value list>.

   (*IVL*)

   is equivalent to the <table value constructor>:

   (VALUES *IVL*)

2) Let *RVC* be the <row value constructor> and let *IPV* be the <in predicate value>.

3) The expression

   RVC NOT IN *IPV*

   is equivalent to

   NOT (*RVC* IN *IPV*)

4) The expression

   RVC IN *IPV*

   is equivalent to

   RVC = ANY *IPV*

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain a <value expression> in an <in value list> that is not a <value specification>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.5 <like predicate>

Function

Specify a pattern-match comparison.

Format

```
<like predicate> ::=
       <match value> [ NOT ] LIKE <pattern>
         [ ESCAPE <escape character> ]

<match value> ::= <character value expression>

<pattern> ::= <character value expression>

<escape character> ::= <character value expression>
```

Syntax Rules

1) The data types of <match value>, <pattern>, and <escape character> shall be character string. <match value>, <pattern>, and <escape character> shall be comparable.

2) Let *M* be the result of the <character value expression> of the <match value>, let *P* be the result of the <character value expression> of the <pattern>, and let *E* be the result of the <character value expression> of the <escape character> if one is specified.

3) ''*M* NOT LIKE *P*'' is equivalent to ''NOT (*M* LIKE *P*)''.

4) Case:

   a) If <escape character> is not specified, then the collating sequence used for the <like predicate> is determined by Table 3, "Collating sequence usage for comparisons", taking <match value> as comparand 1 and <pattern> as comparand 2.

   b) Otherwise, let *C1* be the coercibility attribute and collating sequence of the <match value>, and *C2* be the coercibility attribute and collating sequence of the <pattern>. Let *C3* be the resulting coercibility attribute and collating sequence as determined by Table 2, "Collating coercibility rules for dyadic operators", taking *C1* as the operand 1 coercibility and *C2* as the operand 2 coercibility. The collating sequence used for the <like predicate> is determined by Table 3, "Collating sequence usage for comparisons", taking *C3* as the coercibility attribute and collating sequence of comparand 1 and <escape character> as comparand 2.

5) If all of the values *M*, *P*, and *E* that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) Case:

   a) If ESCAPE is not specified and either *M* or *P* are null values, then

      Case:

      i) If the result has the general null class, then the result of

         M LIKE *P*

         is the general null value.

      ii) Otherwise, the result of

         M LIKE *P*

         is the minimum of the null values of *M* and *P*.

   b) If ESCAPE is specified and one or more of *M*, *P* and *E* are null values, then

      Case:

      i) If the result has the general null class, then the result of

         M LIKE *P* ESCAPE *E*

         is the general null value.

      ii) Otherwise the result of

         M LIKE *P* ESCAPE *E*

         is the minimum of the null values of *M*, *P*, and *E*.

   Note: If none of *M*, *P*, and *E* (if present) are null values, then the result is either true or false.

2) Case:

   a) If an <escape character> is specified, then:

      i) If the length in characters of *E* is not equal to 1, then an exception condition is raised: *data exception—invalid escape character*.

      ii) If there is not a partitioning of the string *P* into substrings such that each substring has length 1 or 2, no substring of length 1 is the escape character *E*, and each substring of length 2 is the escape character *E* followed by either the escape character *E*, an <underscore> character, or the <percent> character, then an exception condition is raised: *data exception—invalid escape sequence*.

      If there is such a partitioning of *P*, then in that partitioning, each substring with length 2 represents a single occurrence of the second character of that substring. Each substring with length 1 that is the <underscore> character represents an arbitrary character specifier. Each substring with length 1 that is the <percent> character represents an arbitrary string specifier. Each substring with length 1 that is neither the <underscore> character nor the <percent> character represents the character that it contains.

   b) If an <escape character> is not specified, then each <underscore> character in *P* represents an arbitrary character specifier, each <percent> character in *P* represents an arbitrary string specifier, and each character in *P* that is neither the <underscore> character nor the <percent> character represents itself.

3) The string *P* is a sequence of the minimum number of substring specifiers such that each <character representation> of *P* is part of exactly one substring specifier. A substring specifier is an arbitrary character specifier, an arbitrary string specifier, or any sequence of <character representation>s other than an arbitrary character specifier or an arbitrary string specifier.

4) If either *M* or *P* is a null value, then the result of ''*M* LIKE *P*'' is the general null value. If *M* and *P* are non-null values, then ''*M* LIKE *P*'' is either true or false.

5) Case:

   a) If *M* and *P* are character strings whose lengths are variable and if the lengths of both *M* and *P* are 0, then

      M LIKE *P*

      is true.

   b) The <predicate>

      M LIKE *P*

      is true if there exists a partitioning of *M* into substrings such that:

      i) A substring of *M* is a sequence of 0 or more contiguous <character representation>s of *M* and each <character representation> of *M* is part of exactly one substring.

      ii) If the *i*-th substring specifier of *P* is an arbitrary character specifier, the *i*-th substring of *M* is any single <character representation>.

      iii) If the *i*-th substring specifier of *P* is an arbitrary string specifier, then the *i*-th substring of *M* is any sequence of 0 or more <character representation>s.

      iv) If the *i*-th substring specifier of *P* is neither an arbitrary character specifier nor an arbitrary string specifier, then the *i*-th substring of *M* is equal to that substring specifier according to the collating sequence of the <like predicate>, without the appending of <space> characters to *M*, and has the same length as that substring specifier.

      v) The number of substrings of *M* is equal to the number of substring specifiers of *P*.

   c) Otherwise,

      M LIKE *P*

      is false.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) The <match value> shall be a <column reference>.

b) A <pattern> shall be a <value specification>.

c) An <escape character> shall be a <value specification>.

8.6 <similar predicate>

Function

Specify a character string similarity by means of a regular expression.

Format

```
<similar predicate> ::=
```

ANSI Only--SQL3

```
      <match value> [ NOT ] SIMILAR TO <similar pattern>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
      <match value> [ NOT ] SIMILAR TO <pattern>

        [ ESCAPE <escape character> ]
```

ANSI Only--SQL3

```
<similar pattern> ::= <character value expression>

<regular expression> ::=
         <regular term>
       | <regular expression> <vertical bar> <regular term>

<regular term> ::=
         <regular factor>
       | <regular term> <regular factor>

<regular factor> ::=
         <regular primary>
       | <regular primary> <asterisk>
       | <regular primary> <plus sign>

<regular primary> ::=
```

ANSI Only--SQL3

```
        <character specifier>
       | <percent>
       | <regular character set>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
        <character representation>
       | <character set>
```

```
        | <left paren> <regular expression> <right paren>
```

ANSI Only--SQL3

```
<character specifier> ::=   <non-escaped character> | <escaped character>

<non-escaped character> ::= !! See the Syntax Rules

<escaped character> ::= !! See the Syntax Rules

<regular character set> ::=
          <underscore>
        | <left bracket> <character enumeration> <right bracket>
        | <left bracket> <circumflex> <character enumeration> <right bracket>
        | <left bracket> <colon> <regular character set identifier> <colon> <right bracket>


<character enumeration> ::=
          <character specifier>
        | <character specifier> <minus sign> <character specifier>
        | <character specifier>...

<regular character set identifier> ::= <identifier>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
<character set> ::=
          <left bracket> <character enumeration> <right bracket>
        | <left bracket> <circumflex> <character enumeration> <right bracket>
        | <left bracket> <colon> <character set specifier> <colon> <right bracket>

<character enumeration> ::=
          <character representation>
        | <character representation> <minus sign> <character representation>
        | <character enumeration>...

<character set specifier> ::= <identifier>
```

Syntax Rules

1) The result of the <character value expression> of <escape character> shall have length 1.

2) The data types of <match value>, <pattern>, and <escape character> shall be character string. <match value>, <pattern>, and <escape character> shall be comparable.

3) The value of the <identifier> that is a
   ANSI <regular character set identifier>
   ISO <character set specifier>
   shall be either ALPHA, UPPER, LOWER, DIGIT, or ALNUM.

****Editor's Note****
The set of <identifier>s for <regular character set identifier>s could profitable be enlarged to support additional sorts of characters (*e.g.*, ideographs, syllables, *etc*.), as a result of internationalization work subh as that going on in SC22/WG20. See Opportunity **|268|**.

ANSI Only–SQL3

4) Case:

   a) If <escape character> is not specified, then the collating sequence used for the <similar predicate> is determined by Table 3, "Collating sequence usage for comparisons", taking <match value> as comparand 1 and <similar pattern> as comparand 2.

   b) Otherwise, let *C1* be the coercibility attribute and collating sequence of the <match value>, and *C2* be the coercibility attribute and collating sequence of the <similar pattern>. Let *C3* be the resulting coercibility attribute and collating sequence as determined by Table 2, "Collating coercibility rules for dyadic operators", taking *C1* as the operand 1 coercibility and *C2* as the operand 2 coercibility. The collating sequence used for the <simlar predicate> is determined by Table 3, "Collating sequence usage for comparisons", taking *C3* as the coercibility attribute and collating sequence of comparand 1 and <escape character> as comparand 2.

   It is implementation defined, whether all, some, or no collating sequences other than the default collating sequence for the character set of the <match value> can be used as the collating sequence of the <similar predicate>.

ISO Only—caused by ANSI changes not yet considered by ISO

5) The collating sequence of the <similar predicate> is determined in the same manner as described in Subclause 8.2, "<comparison predicate>", with <match value> taking the role of *M* and <pattern> taking the role of *P*.

ANSI Only–SQL3

6) A <non-escaped character> is any single character from the character set of the <similar pattern> that is not a <left bracket>, <right bracket>, <left paren>, <right paren>, <vertical bar>, <circumflex>, <minus sign>, <plus sign>, <asterisk>, <underscore>, <percent>, or the character specified by the result of the <character value expression> of <escape character>. A <character specifier> that is a <non-escaped character> represents itself.

7) An <escaped character> is a sequence of two characters: the character specified by the result of the <character value expression> of <escape character>, followed by a second character that is a <left bracket>, <right bracket>, <left paren>, <right paren>, <vertical bar>, <circumflex>, <minus sign>, <plus sign>, <asterisk>, <underscore>, <percent>, or the character specified by the result of the <character value expression> of <escape character>. A <character specifier> that is an <escaped character> represents its second character.

8) If the result *E* of the <character value expression> of <escape character> is one of <left bracket>, <right bracket>, <left paren>, <right paren>, <vertical bar>, <circumflex>, <minus sign>, <plus sign>, <asterisk>, <underscore> or <percent>, then *E* shall not be specified in the <regular expression> except as an <escaped character>.

9) If the result of the <character value expression> of <escape character> is a <colon>, then the <regular expression> shall not contain a <regular character set identifier>.

10) A <character enumeration> shall not be specified in a way that both its first and its last <character specifier>s are <non-escaped character>s that are <colon>s.

---

11) If all of the values of <match value>, <pattern>, and <escape character> that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) Let *M* be the result of the <character value expression> of the <match value> and let *P* be the result of the <character value expression> of the <pattern>.

2) *M* NOT SIMILAR TO *P*

is equivalent to

NOT (*M* SIMILAR TO *P*)

ISO Only—caused by ANSI changes not yet considered by ISO

---

3) Case:

a) If an <escape character> is specified, then:

i) Let *E* be the result of the <character value expression> of the <escape character>.

ii) If there is not a partitioning of the string *P* into substrings such that each substring has length 1 or 2, no substring of length 1 is the escape character *E*, and each substring of length 2 is the escape character *E* followed by either the escape character *E*, <underscore>, <percent>, <asterisk>, <plus sign>, <minus sign>, <vertical bar>, <left bracket>, <right bracket>, <left paren>, <circumflex>, or <right paren>, then an exception condition is raised: *data exception—invalid escape character*.

Otherwise, in that partitioning of *P*, each substring shall be as follows:

1) Each substring with length 2 represents a single instance of the second character of that substring.

2) Each substring with length 1 that is <underscore> represents an arbitrary character specifier.

3) Each substring with length 1 that is <percent> represents an arbitrary string specifier.

4) Each substring with length 1 that is <asterisk> represents an asterisk specifier.

5) Each substring with length 1 that is <plus sign> represents a plus specifier.

6) Each substring with length 1 that is <minus sign> represents a minus specifier.

7) Each substring with length 1 that is <vertical bar> represents a vertical bar specifier.

8) Each substring with length 1 that is <left bracket> represents a left square bracket specifier.

9) Each substring with length 1 that is <right bracket> represents a right square bracket specifier.

10) Each substring with length 1 that is <left paren> represents a left parenthesis specifier.

11) Each substring with length 1 that is <right paren> represents a right parenthesis specifier.

12) Each substring with length 1 that is <circumflex> represents a circumflex specifier.

13) Each substring with length 1 that is neither <percent>, <underscore>, <asterisk>, <plus sign>, <minus sign>, <vertical bar>, <left bracket>, <right bracket>, <left paren>, <circumflex>, nor <right paren> is a character specifier that represents the character that it contains.

b) If an <escape character> is not specified, then each character in *P* shall be as follows:

i) Each <underscore> in *P* represents an arbitrary character specifier.

ii) Each <percent> in *P* represents an arbitrary string specifier.

iii) Each <asterisk> in *P* represents an asterisk specifier.

iv) Each <plus sign> in *P* represents a plus specifier.

v) Each <minus sign> in *P* represents a minus specifier.

vi) Each <vertical bar> in *P* represents a vertical bar specifier.

vii) Each <left bracket> in *P* represents a left square bracket specifer.

viii) Each <right bracket> in *P* represents a right square bracket specifier.

ix) Each <left paren> in *P* represents a left parenthesis specifier.

x) Each <right paren> in *P* represents a right parenthesis specifier.

xi) Each <circumflex> in *P* represents a circumflex specifier.

xii) Each character in *P* that is neither <underscore>, <percent>, <asterisk>, <plus sign>, <minus sign>, <vertical bar>, <left bracket>, <right bracket>, <left paren>, <circumflex>, nor <right paren> represents itself.

4) The string *R* is a sequence of the minimum number of substring specifiers such that each <character representation> of *P* is part of exactly one substring specifier. A substring specifier is an arbitrary character specifier, an arbitrary string specifier, an asterisk specifier, a plus specifier, a minus specifier, a vertical bar specifier, a left square bracket specifier, a right square bracket specifier, a left parenthesis specifier, a right parenthesis specifier, a circumflex specifier, or any sequence of character specifiers.

ANSI Only–SQL3

5) If the result of the <character value expression> of the <similar pattern> is not an empty string and does not have the format of a <regular expression>, then an exception condition is raised: *data exception—invalid regular expression*.

6) Case:

   a) If ESCAPE is not specified, then if either or both of *M* and *P* are null values, then

      Case:

      i) If the result has the general null class, then the result of

         M SIMILAR TO *P*

         is the general null value.

      ii) Otherwise, the result of

         M SIMILAR TO *P*

         is the minimum of the null values of *M* and *P*.

   b) If ESCAPE is specified, then if one or more of *M*, *P*, and *E* are null values, then

      Case:

      i) If the result has the general null class, then the result of

         M SIMILAR TO *P* ESCAPE *E*

         is the general null value.

      ii) Otherwise, the result of

         M SIMILAR TO *P* ESCAPE *E*

         is the minimum of the null values of *M*, *P*, and *E*.

Note: If none of *M*, *P*, and *E* (if present) are null values, then the result is either true or false.

ANSI Only–SQL3

7) The set of characters in a <character enumeration> is defined as

   a) If the enumeration is specified in the form ''<character specifier> <minus sign> <character specifier>'', then the set of all characters that collate greater than or equal to the character represented by the left <character specifier> and less than or equal to the character represented by the right <character specifier>, according to the collating sequence of the pattern *P*.

   b) Otherwise, the set of all characters that the <character specifier>s in the <character enumeration> represent.

8) Let *R* be the result of the <character value expression> of the <similar pattern>. The regular language L(*R*) of the <similar pattern> is a (possibly infinite) set of strings. It is defined recursively for well-formed <regular expression>s *Q*, *Q1*, and *Q2* by the following rules:

   a) L(*Q1* <vertical bar> *Q2*)

      is the union of L(*Q1*) and L(*Q2*)

   b) L(*Q* <asterisk>)

      is the set of all strings that can be constructed by concatenating zero or more strings from L(*Q*).

   c) L(*Q* <plus sign>)

      is the set of all strings that can be constructed by concatenating one or more strings from L(*Q*).

   d) L(<character specifier>)

      is a set that contains a single string of length 1 with the character that the <character specifier> represents

   e) L(<percent>)

      is the set of all strings of any length (zero or more) from the character set of the pattern *P*.

   f) L(<left paren> *Q* <right paren>)

      is equal to L(*Q*)

   g) L(<underscore>)

      is the set of all strings of length 1 from the character set of the pattern *P*.

   h) L(<left bracket> <character enumeration> <right bracket>)

      is the set of strings of length 1 with character values as defined from the <character enumeration>.

   i) L(<left bracket> <circumflex> <character enumeration> <right bracket>)

      is the set of strings of length 1 with character values from the character set of the pattern *P* that are not contained in the <character enumeration>.

   j) L(<left bracket> <colon> ALPHA <colon> <right bracket>)

      is the set of character strings of length 1 that are <simple Latin letter>s.

   k) L(<left bracket> <colon> UPPER <colon> <right bracket>)

is the set of character strings of length 1 that are <simple Latin uppercase letter>s.

l) L(<left bracket> <colon> LOWER <colon> <right bracket>)

is the set of character strings of length 1 that are <simple Latin lower case letter>s.

m) L(<left bracket> <colon> DIGIT <colon> <right bracket>)

is the set of character strings of length 1 that are <digit>s.

n) L(<left bracket> <colon> ALNUM <colon> <right bracket>)

is the set of character strings of length 1 that are <simple Latin letter>s or <digit>s.

o) L(*Q1* || *Q2*)

is the set of all strings that can be constructed by concatenating one element of L(*Q1*) and one element of L(*Q2*).

p) L(*Q*)

is the set of the empty string, if *Q* is an empty regular expression.

9) The <similar predicate>

M SIMILAR TO *P*

is true, if there exists at least one element *X* of L(*R*) that is equal to *M* according to the collating sequence of the <similar predicate>; otherwise, it is false.

Note: The <similar predicate> is defined differently from equivalent forms of the LIKE predicate. In particular, blanks at the end of a pattern and collating sequences are handled differently.

ISO Only—caused by ANSI changes not yet considered by ISO

10) If *M* and *P* are character strings whose lengths are variable and if the lengths of both *M* and *P* are 0, then ''*M* SIMILAR TO *P*'' is true.

11) For a character string *M* and a regular expression *R*, ''*M*' SIMILAR TO '*R*'' is defined as follows:

a) '*M*' SIMILAR TO ' W_1' || '|' || 'W_2'

is true if and only if

'*M*' SIMILAR TO 'W_1' OR '*M*' SIMILAR TO 'W_2'

is true.

b) 'X_1' || 'X_2' SIMILAR TO 'W_1' || 'W_2'

is true if and only if

'X_1' SIMILAR TO 'W_1' AND 'X_2' SIMILAR TO 'W_2'

is true.

c) For a non-negative integer *n*,

'X_1' || . . . || 'X_n' SIMILAR TO '*R*' || '*'

is true if and only if

'X_1' SIMILAR TO 'R' AND . . . AND 'X_n' SIMILAR TO 'R'

is true.

d) For a positive integer n,

'X_1' || . . . || 'X_n' SIMILAR TO 'R' || '+'

is true if and only if

'X_1' SIMILAR TO 'R' AND . . . AND 'X_n' SIMILAR TO 'R'

is true.

e) For a character C,

'C' SIMILAR TO 'R'

is true if and only if

'C' = 'R'

is true.

f) For a character C,

'C' SIMILAR TO '[' || 'C' || ']'

is true if and only if

'C' = 'C'

is true.

g) For characters C, C_1, and C_2,

'C' SIMILAR TO '[' || 'C_1' || '–' || 'C_2' ']'

is true if and only if

'C' BETWEEN 'C_1' AND 'C_2'

is true.

h) For <character enumeration>s E_1 and E_2,

'C' SIMILAR TO '[' || 'E_1' || 'E_2' || ']'

is true if and only if

'C' SIMILAR TO '[' || 'E_1' || ']' OR 'C' SIMILAR TO '[' || 'E_2' || ']'

is true.

i) For a <character enumeration> E,

'C' SIMILAR TO '[' || '^' || 'E' || ']'

is true if and only if

NOT ('C' SIMILAR TO '[' || 'E' || ']')

is true.

j) For a character *C*,

'*C*' SIMILAR TO '[:ALPHA:]'

is true if and only if *C* is a <simple Latin letter>.

k) For a character *C*,

'*C*' SIMILAR TO '[:UPPER:]'

is true if and only if *C* is a <simple Latin upper case letter>.

l) For a character *C*,

'*C*' SIMILAR TO '[:LOWER:]'

is true if and only if *C* is a <simple Latin lower case letter>.

m) For a character *C*,

'*C*' SIMILAR TO '[:DIGIT:]'

is true if and only if *C* is a <digit>.

n) For a character *C*,

'*C*' SIMILAR TO '[:ALNUM:]'

is true if and only if *C* is a <simple Latin letter> or a <digit>.

o) '*M*' SIMILAR TO '(' || '*R*' || ')'

is equal to

'*M*' SIMILAR TO '*R*'

Leveling Rules

1) The following restrictions apply for Full SQL:

a) Conforming Full SQL language shall contain no <similar predicate>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

8.7 <null predicate>

Function

Specify a test for a null value.

Format

```
<null predicate> ::= <row value constructor> IS [ NOT ] NULL
          [ <left paren> <null values specification> <right paren> ]

<null values specification> ::= <asterisk> | <null state>
```

Syntax Rules

1) The null class of a <value expression> is defined in Subclause 6.14, "<value expression>".

2) If either no <value expression> of the <row value constructor> has a defined null class or any two <value expression>s of the <row value constructor> have different defined null classes, then <null state> shall not be specified. Otherwise, <null state> shall be a valid null state of the unique defined null class.

Access Rules

None.

General Rules

1) Let *R* be the value of the <row value constructor>.

2) If all the values in *R* are a null value, then ''*R* IS NULL'' is true; otherwise, it is false.

3) If none of the values in *R* are a null value, then ''*R* IS NOT NULL'' is true; otherwise, it is false.

Note: For all *R*, ''*R* IS NOT NULL'' has the same result as ''NOT *R* IS NULL'' if and only if *R* is of degree 1. Table 14, "<null predicate> semantics", specifies this behavior.

Table 14—<null predicate> semantics

Expression	*R* IS NULL	*R* IS NOT NULL	NOT *R* IS NULL	NOT *R* IS NOT NULL
degree 1: null	*true*	*false*	*false*	*true*
degree 1: not null	*false*	*true*	*true*	*false*
degree > 1: all null	*true*	*false*	*false*	*true*
degree > 1: some null	*false*	*false*	*true*	*true*
degree > 1: none null	*false*	*true*	*true*	*false*

4) Case:

   a) If <null class> is not specified, then:

      i) ''*R* IS NULL'' is true if and only if all the values in *R* are the general null value; otherwise, it is false.

      ii) ''*R* IS NOT NULL'' is true if and only if all the values in *R* are non-null or have a null value that is not the general null value; otherwise, it is false.

   b) If <null class> is specified as '' *''', then:

      i) ''*R* IS NULL(*)'' is true if and only if all the values in *R* are some null value; otherwise, it is false.

      ii) ''*R* IS NOT NULL(*)'' is true if and only if all the values in *R* are non-null; otherwise, it is false.

   c) If <null class> is specified as ''<null state>'', then:

      i) Let *C* be the null state specified by <null state>.

      ii) ''*R* IS NULL (*C*)'' is true if and only if all the values in *R* are a null value with a null state of *C*; otherwise, it is false.

      iii) ''*R* IS NOT NULL (*C*)'' is true if and only if all the values in *R* are non-null or have a null value with a null state other than that of *C*; otherwise, it is false.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <null predicate> shall contain no <null values specification>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <row value constructor> shall be a <column reference>.

8.8 <quantified comparison predicate>

Function

Specify a quantified comparison.

Format

```
<quantified comparison predicate> ::=
       <row value constructor> <comp op> <quantifier> <table subquery>

<quantifier> ::= <all> | <some>

<all> ::= ALL

<some> ::= SOME | ANY
```

Syntax Rules

1) The <row value constructor> shall be of the same degree as the result of the

ANSI Only–SQL3

<table subquery>, or the <row value constructor> shall have a degree of one and the data type of the <row value constructor> shall be comparable to the data type of the objects that are the result of the

<table subquery>.

ANSI Only–SQL3

2) Case:

   a) If the <row value constructor> has a degree of one and the <data type> of the single value of the <row value constructor> is comparable to the <data type> of the objects in *T*, then the implied <comparison predicate> ''*R* <comp op> *RT*'' compares that single value to the object corresponding to a row in *T*.

   b) Otherwise, ''*R* <comp op> *RT*'' compares the row *R* to a row *RT* in *T*.

3) The data types of the values of the <row value constructor> shall be respectively comparable to those of the corresonding columns of the <table subquery>.

4) The collating sequence for each pair of respective values in the <quantified comparison predicate> is determined in the same manner as described in Subclause 8.2, "<comparison predicate>".

Access Rules

None.

General Rules

1) Let *R* be the result of the <row value constructor> and let *T* be the result of the <table subquery>.

2) The result of ''*R* <comp op> <quantifier> *T*'' is derived by the application of the implied <comparison predicate> ''*R* <comp op> *RT*'' to every row *RT* in *T*:

   Case:

   a) If *T* is empty or if the implied <comparison predicate> is true for every row *RT* in *T*, then ''*R* <comp op> <all> *T*'' is true.

   b) If the implied <comparison predicate> is false for at least one row *RT* in *T*, then ''*R* <comp op> <all> *T*'' is false.

   c) If the implied <comparison predicate> is true for at least one row *RT* in *T*, then ''*R* <comp op> <some> *T*'' is true.

   d) If *T* is empty or if the implied <comparison predicate> is false for every row *RT* in *T*, then ''*R* <comp op> <some> *T*'' is false.

   e) If ''*R* <comp op> <quantifier> *T*'' is neither true nor false, then it is the general null value.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.9 <exists predicate>

Function

Specify a test for a non-empty set.

Format

```
<exists predicate> ::= EXISTS <table subquery>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *T* be the result of the <table subquery>.

2) If the cardinality of *T* is greater than 0, then the result of the <exists predicate> is true; otherwise, the result of the <exists predicate> is false.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.10 <unique predicate>

Function

Specify a test for the absence of duplicate rows.

Format

```
<unique predicate> ::= UNIQUE <table subquery>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *T* be the result of the <table subquery>.

2) If there are no two rows in *T* such that the value of each column in one row is non-null and is equal to the value of the corresponding column in the other row according to Subclause 8.2, "<comparison predicate>", then the result of the <unique predicate> is true; otherwise, the result of the <unique predicate> is false.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <unique predicate>.

8.11 <match predicate>

Function

Specify a test for matching rows.

Format

```
<match predicate> ::=
       <row value constructor> MATCH [ UNIQUE ] [ PARTIAL | FULL ] <table subquery>
```

Syntax Rules

1) The <row value constructor> shall be of the same degree as the

ANSI Only–SQL3

<table subquery>, or the <row value constructor> shall have a degree of one and the data type of the <row value constructor> shall be comparable to the data type of the objects that are the result of the

<table subquery>.

2) The data types of the values of the <row value constructor> shall be respectively comparable to those of the corresponding columns of the <table subquery>.

3) The collating sequence for each pair of respective values in the <match predicate> is determined in the same manner as described in Subclause 8.2, "<comparison predicate>".

Access Rules

None.

General Rules

1) Let *R* be the <row value constructor>.

ANSI Only–SQL3

Let *T* be the result of the <table subquery>.

2) Case:

   a) If the <row value constructor> *R* has a degree of one and the <data type> of the single value of the <row value constructor> is comparable to the <data type> of the objects in *T*, then a comparison of *R* with *RTi*, a row of *T*, compares that single value to the object corresponding to the row *RTi* in *T*.

   b) Otherwise, a comparison of *R* with *RTi* compares the row *R* to a row *RTi* in *T*.

3) If neither PARTIAL nor FULL is specified, then

Case:

a) If some value in *R* is a null value, then the <match predicate> is true.

b) If no value in *R* is a null value, then

Case:

i) If UNIQUE is not specified and there exists a (possibly non-unique) row RT_i of the <table subquery> such that

$R = RT_i$

then the <match predicate> is true.

ii) If UNIQUE is specified and there is a unique row RT_i of the <table subquery> such that

$R = RT_i$

then the <match predicate> is true.

iii) Otherwise, the <match predicate> is false.

4) If PARTIAL is specified, then

Case:

a) If all values in *R* are a null value, then the <match predicate> is true.

b) Otherwise,

Case:

i) If UNIQUE is not specified and there exists a (possibly non-unique) row RT_i of the <table subquery> such that each non-null value of *R* equals its corresponding value in RT_i, then the <match predicate> is true.

ii) If UNIQUE is specified and there is a unique row RT_i of the <table subquery> such that each non-null value of *R* equals its corresponding value in RT_i, then the <match predicate> is true.

iii) Otherwise, the <match predicate> is false.

5) If FULL is specified, then

Case:

a) If all values in *R* are a null value, then the <match predicate> is true.

b) If no values in *R* are a null value, then

Case:

i) If UNIQUE is not specified and there exists a (possibly non-unique) row RT_i of the <table subquery> such that

$R = RT_i$

then the <match predicate> is true.

ii) If UNIQUE is specified and there exists a unique row RT_i of the <table subquery> such that

$R = RT_i$

then the <match predicate> is true.

iii) Otherwise, the <match predicate> is false.

c) Otherwise, the <match predicate> is false.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <match predicate>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

8.12 <overlaps predicate>

Function

Specify a test for an overlap between two events.

Format

```
<overlaps predicate> ::=
       <row value constructor 1> OVERLAPS <row value constructor 2>

<row value constructor 1> ::= <row value constructor>

<row value constructor 2> ::= <row value constructor>
```

Syntax Rules

1) The degree of <row value constructor 1> and <row value constructor 2> shall both be 2.

2) The data type of the first column of each <row value constructor> shall be a datetime data type and the first column of each <row value constructor> shall be comparable.

   Note: Two datetimes are comparable only if they have the same <datetime field>s; see Subclause 4.7.1, "Datetimes".

3) The data type of the second column of each <row value constructor> shall be a datetime data type or INTERVAL.

   Case:

   a) If the data type is INTERVAL, then the precision of the data type shall be such that the interval can be added to the datetime data type contained in the first column of the <row value constructor>.

   b) If the data type is a datetime data type, then it shall be comparable with the datetime data type contained in the first column of the <row value constructor>.

Access Rules

None.

General Rules

1) Let *D1* be the value of the first column of <row value constructor 1> and *D2* be the value of the first column of <row value constructor 2>.

2) Case:

   a) If the data type of the second column of <row value constructor 1> is a datetime data type, then let *E1* be the value of the second column of <row value constructor 1>.

   b) If the data type of the second column of <row value constructor 1> is INTERVAL, then let *I1* be the value of the second column of <row value constructor 1>. Let *E1* = *D1* + *I1*.

3) If *D1* is the null value or if *E1* < *D1*, then let *S1* = *E1* and let *T1* = *D1*. Otherwise, let *S1* = *D1* and let *T1* = *E1*.

4) Case:

   a) If the data type of the second column of <row value constructor 2> is a datetime data type, then let *E2* be the value of the second column of <row value constructor 2>.

   b) If the data type of the second column of <row value constructor 2> is INTERVAL, then let *I2* be the value of the second column of <row value constructor 2>. Let *E2* = *D2* + *I2*.

5) If *D2* is the null value or if *E2* < *D2*, then let *S2* = *E2* and let *T2* = *D2*. Otherwise, let *S2* = *D2* and let *T2* = *E2*.

6) The result of the <overlaps predicate> is the result of the following expression:

   (*S1* > *S2* AND NOT (*S1* >= *T2* AND *T1* >= *T2*))
   OR
   (*S2* > *S1* AND NOT (*S2* >= *T1* AND *T2* >= *T1*))
   OR
   (*S1* = *S2* AND (*T1* <> *T2* OR *T1* = *T2*))

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <overlaps predicate>.

8.13 <quantified predicate>

Function

Specify a quantified predicate.

Format

```
<quantified predicate> ::=
         <existential clause> <left paren> <search condition> <right paren>
       | <universal clause> <left paren> <search condition> <right paren>
       | <quantified comparison predicate>

<existential clause> ::=
       FOR SOME <table reference list>

<universal clause> ::=
       FOR ALL <table reference list>

<table reference list> ::=
       <table reference> [ { <comma> <table reference> }... ]
```

Syntax Rules

1) Case:

   a) If the <quantified predicate> immediately contains an <existential clause>, then all Syntax Rules of Subclause 7.7, "<from clause>", apply to the <existential clause>, replacing ''<from clause>'' with ''<existential clause>'' throughout, except that Syntax Rule 1)a) is replaced with:

      If a <table reference> *TR* is contained in an <existential clause> *FC* with no intervening <derived table>, then the scope clause *SC* of *TR* is the innermost <quantified predicate> that contains *FC*. The scope of the exposed <correlation name> or exposed <table name> of *TR* is *SC*, together with the <join condition> of all <joined table>s contained in *SC* that contains *TR*.

   b) If the <quantified predicate> immediately contains a <universal clause>, then all Syntax Rules of Subclause 7.7, "<from clause>", apply to the <existential clause>, replacing ''<from clause>'' with ''<universal clause>'' throughout, except that Syntax Rule 1)a) is replaced with:

      If a <table reference> *TR* is contained in a <universal clause> *FC* with no intervening <derived table>, then the scope clause *SC* of *TR* is the innermost <quantified predicate> that contains *FC*. The scope of the exposed <correlation name> or exposed <table name> of *TR* is *SC*, together with the <join condition> of all <joined table>s contained in *SC* that contains *TR*.

Access Rules

1) Let *T* be any table that is referenced in an <existential clause> or <universal clause>. If *T* is not a declared temporary table or a temporary view, then then the applicable privileges shall include SELECT for at least one column of *T*.

8.13 <quantified predicate>

General Rules

1) Case:

   a) If the <quantified predicate> immediately contains an <existential clause>, then all General Rules of Subclause 7.7, "<from clause>", apply to the <existential clause>, replacing each occurrence of ''<from clause>'' with ''<existential clause>''.

   b) If the <quantified predicate> immediately contains a <universal clause>, then all General Rules of Subclause 7.7, "<from clause>", apply to the <universal clause>, replacing each occurrence of ''<from clause>'' with ''<universal clause>''.

2) Case:

   a) If the <quantified predicate> immediately contains an <existential clause>, then let *T* be the result of this <existential clause>. The result of the <quantified predicate> is:

      i) false if *T* is empty or if the <search condition> is false for every row of *T*;

      ii) true if there exists any row of *T* for which the <search condition> is true;

      iii) unknown otherwise.

   b) If the <quantified predicate> immediately contains a <universal clause>, then let *T* be the result of this <universal clause>. The result of the <quantified predicate> is:

      i) true if *T* is empty or if the <search condition> is true for every row of *T*;

      ii) false if there exists any row of *T* for which the <search condition> is false;

      iii) unknown otherwise.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <quantified predicate>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.14 <there is predicate>

Function

Specify an existentially quantified predicate.

Format

```
<there is predicate> ::=
       <left paren> <there is clause> <where clause> <right paren>

<there is clause> ::=
       THERE IS <table reference list>
```

Syntax Rules

1) All Syntax Rules of Subclause 7.7, "<from clause>", apply to the <there is predicate>, replacing ''<from clause>'' with ''<there is clause>'' and ''<subquery>, <query specification>, or <select statement: single row>'' with ''<there is predicate>''.

Access Rules

None.

General Rules

1) All General Rules of Subclause 7.7, "<from clause>", apply to the <there is clause>, replacing ''<from clause>'' with ''<there is clause>''.

2) Let *T* be the result of the <there is clause>. The result of the <there is predicate> is:

   a) false if *T* is empty or if the <where clause> is false for every row of *T*.

   b) true if there exists any row of *T* for which the <where clause> is true.

   c) unknown otherwise.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <there is predicate>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.15 <distinct predicate>

Function

Specify a test of whether two row values are distinct

Format

```
<distinct predicate> ::=
       <row value constructor 1> IS DISTINCT FROM <row value constructor 2>

<row value constructor 1> ::= <row value constructor>

<row value constructor 2> ::= <row value constructor>
```

Syntax Rules

1) The two <row value constructor>s shall be of the same degree.

2) Let *respective values* be values with the same ordinal position.

3) The data types of the respective values of the two <row value constructor>s shall be comparable.

4) Let *X* be a value in the first <row value constructor> and let *Y* be the respective value in the second <row value constructor>.

Access Rules

None.

General Rules

1) The result of the <distinct predicate> is either true or false.

2) Two <row value constructor>s are distinct if, for any pair of respective values *X* and *Y*, *X* IS DISTINCT FROM *Y* is true.

ANSI Only–SQL3

3) Case:

   a) If the data type of *X* or *Y* is a set or multiset type, then ''*X* IS DISTINCT FROM *Y*'' is effectively computed as follows:

      i) Let *TEMPX* be a copy of *X*; let *TEMPY* be a copy of *Y*.

      ii) For each element of *TEMPX*, if that element is not distinct from an element in *TEMPY*, then delete that element from *TEMPX* and that element in *TEMPY*.

      iii) If *TEMPX* is empty and *TEMPY* is empty, then ''*X* IS DISTINCT FROM *Y*'' is false. Otherwise, ''*X* IS DISTINCT FROM *Y*'' is true.

b) If the data type of *X* or *Y* is a list type, then "*X* IS DISTINCT FROM *Y*" is effectively computed as follows:

i) Let *NX* be the number of elements in *X*; let *NY* be the number of elements in *Y*.

ii) Let EX_i be the *i*-th element of *X*; let E Y_i be the *i*-th element of *Y*.

iii) If *NX* is not equal to *NY*, then "*X* IS DISTINCT FROM *Y*" is true.

iv) If *NX* equals zero and *NY* equals zero, then "*X* IS DISTINCT FROM *Y*" is false.

v) If "EX_i IS DISTINCT FROM EY_i" is false for all *i* from *i=1* to *i=NX*, then "*X* IS DISTINCT FROM *Y*" is false.

vi) Otherwise, "*X* IS DISTINCT FROM *Y*" is true.

c) Otherwise,

Case:

i) ''*X* IS DISTINCT FROM *Y*'' is false if either:

1) *X* and *Y* are the same null value, or

2) *X* = *Y* according to Subclause 8.2, "<comparison predicate>".

ii) Otherwise, ''*X* IS DISTINCT FROM *Y*'' is true.

ISO Only—caused by ANSI changes not yet considered by ISO

4) *X* IS DISTINCT FROM *Y* is false if either:

a) *X* and *Y* are the same null value, or

b) *X* = *Y* according to Subclause 8.2, "<comparison predicate>".

Otherwise, *X* IS DISTINCT FROM *Y* is true.

5) If two <row value constructor>s are not distinct, then they are said to be *duplicates*. If a number of <row value constructor>s are all duplicates of each other, then all except one are said to be *redundant duplicates*.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <distinct predicate>.

8.16 <boolean predicate>

Function

Specify a test for whether a boolean value is *true* , *false* , or *unknown* .

Format

```
<boolean predicate> ::=
       <boolean value expression> [ IS [ NOT ] <truth value> ]

<truth value> ::=
         TRUE
       | FALSE
       | UNKNOWN
```

Syntax Rules

1) If NOT is specified in a <boolean predicate>, then let *BV* be the contained <boolean value expression> and let *TV* be the contained <truth value>. The <boolean predicate> is equivalent to:

   (NOT (*BV* IS *TV*))

Access Rules

None.

General Rules

1) Let *R* be the value of *BV* and let *T* be the value of *TV*.

2) The result of *BV* IS *TV* is derived by the application of the boolean operator ''IS'' to the result derived from the evaluation of *BV*.

3) ''IS'' is defined by Table 15, "Truth table for the IS boolean".

Table 15—Truth table for the IS boolean

IS	TRUE	FALSE	UNKNOWN
true	*true*	*false*	*false*
false	*false*	*true*	*false*
null	*false*	*false*	*true*

Note: The result of a <boolean predicate> is always either *true* or *false* .

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Intermediate SQL language shall not specify a <boolean predicate>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.17 <search condition>

Function

Specify a condition that is true, false, or unknown, depending on the value of a <boolean value expression>.

Format

```
<search condition> ::=
       <boolean value expression>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) When a <search condition> *S* is evaluated against a row of a table, each reference to a column of that table by a <column reference> directly contained in *S* is a reference to the value of that column in that row.

2) Case:

   a) If the result of evaluating the <boolean value expression> is null, then the result of the <search condition> is unknown.

   b) Otherwise, the result of the <search condition> is the result of the <boolean value expression>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <boolean primary> shall not specify a <boolean value expression>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <boolean test> shall not specify a <truth value>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

9 Data assignment rules and function determination

9.1 Retrieval assignment

Function

Specify rules for value assignments that retrieve SQL-data.

Syntax Rules

1) Let *T* and *V* be a *TARGET* and *VALUE* specified in an application of this Subclause.

2) If the data type of *T* is character string, bit string, numeric, an enumerated type, boolean, datetime, interval,
ANSI a collection type,
or an abstract data type, then the data type of *V* shall be a mutually assignable character string type, a bit string type, a numeric type, the same enumerated type, boolean, the same datetime type, a comparable interval type,
ANSI the same collection type,
or the same abstract data type, respectively.

General Rules

1) If *V* is a null
ANSI value and *T* is an <embedded variable specification> or an externally-supplied parameter,
ISO value,
then

Case:

a) If an indicator is specified for *T*, then that indicator is set to

Case:

i) If the null value is the general null value, then −1.

ii) Otherwise, the negative of the position number of the null state.

b) If no indicator is specified for *T*, then an exception condition is raised: *data exception—null value, no indicator parameter*.

ANSI Only–SQL3

2) If *V* is a null value and *T* is not an <embedded variable specification> or an externally-supplied parameter, then *T* is set to *V*.

3) If *V* is not a null value and *T* has an indicator, then

Case:

a) If the data type of *T* is character string or bit string and the length in characters or bits, respectively, *M* of *V* is greater than the length in characters or bits, respectively, of *T*, then the indicator is set to *M*. If *M* exceeds the maximum value that the indicator can contain, then an exception condition is raised: *data exception—indicator overflow*.

b) Otherwise, the indicator is set to 0.

4) If *V* is not a null value, then

Case:

a) If the data type of *T* is fixed-length character string with length in characters *L* and the length in characters of *V* is equal to *L*, then the value of *T* is set to *V*.

b) If the data type of *T* is fixed-length character string with length in characters *L*, and the length in characters of *V* is greater than *L*, then the value of *T* is set to the first *L* characters of *V* and a completion condition is raised: *warning—string data, right truncation*.

c) If the data type of *T* is fixed-length character string with length in characters *L*, and the length in characters *M* of *V* is smaller than *L*, then the first *M* characters of *T* are set to *V*, and the last $L - M$ characters of *T* are set to <space>s.

d) If the data type of *T* is variable-length character string and the length in characters *M* of *V* is not greater than the maximum length in characters of *T*, then the value of *T* is set to *V* and the length in characters of *T* is set to *M*.

e) If the data type of *T* is variable-length character string and the length in characters of *V* is greater than the maximum length in characters *L* of *T*, then the value of *T* is set to the first *L* characters of *V*, the length in characters of *T* becomes *L*, and a completion condition is raised: *warning—string data, right truncation*.

f) If the data type of *T* is fixed-length bit string with length in bits *L* and the length in bits of *V* is equal to *L*, then the value of *T* is set to *V*.

g) If the data type of *T* is fixed-length bit string with length in bits *L* and the length in bits of *V* is greater than *L*, then the value of *T* is set to the first *L* bits of *V* and a completion condition is raised: *warning—string data, right truncation*.

h) If the data type of *T* is fixed-length bit string with length in bits *L* and the length in bits *M* of *V* is smaller than *L*, then the first *M* bits of *T* are set to *V*, the remaining bits of *T* are set to bits each with the value of 0, and a completion condition is raised: *warning—implicit zero-bit padding*.

i) If the data type of *T* is variable-length bit string and the length in bits *M* of *V* is not greater than the maximum length in bits of *T*, then the value of *T* is set to *V* and the length in bits of *T* is set to *M*.

j) If the data type of *T* is variable-length bit string, and the length in bits of *V* is greater than the maximum length in bits *L* of *T*, then the value of *T* is set to the first *L* bits of *V*, the length in bits of *T* is set to *L*, and a completion condition is raised: *warning—string data, right truncation*.

k) If the data type of *T* is numeric and there is an approximation obtained by rounding or truncation of the numerical value of *V* for the data type of *T*, then the value of *T* is set to such an approximation.

If there is no such approximation, then an exception condition is raised: *data exception—numeric value out of range*.

If the data type of *T* is exact numeric, then it is implementation-defined whether the approximation is obtained by rounding or by truncation.

l) If the data type of *T* is datetime and there is a representation of the value of *V* in the data type of *T*, then the value of *T* is set to that representation.

m) If the data type of *T* is interval and there is a representation of the value of *V* in the data type of *T*, then the value of *T* is set to that representation. Otherwise, an exception condition is raised: *data exception—interval field overflow*.

n) If the data type of *T* is an enumerated type, then the value of *T* is set to *V*.

o) If the data type of *T* is boolean, then the value of *T* is set to *V*.

p) If the data type of *T* is an abstract data type, then the value of *T* is set to the value of *V*.

q) If the data type of *T* is object identifier, then the value of *T* is set to *V*.

ANSI Only–SQL3

r) If the data type of *T* is a collection type, then the value of *T* is set to *V*.

9.2 Store assignment

Function

Specify rules for value assignments that store SQL-data or store into external function parameters.

Syntax Rules

1) Let *T* and *V* be a *TARGET* and *VALUE* specified in an application of this Subclause.

2) If *V* is specified by NULL (<null state>), then <null state> shall identify a null state in the null class of *T*.

3) If the data type of *T* is character string, bit string, numeric, an enumerated type, boolean, datetime, interval,
ANSI a collection type,
or an abstract data type, then the data type of *V* shall be character string, bit string, numeric, the same enumerated type, boolean, the same datetime type, a comparable interval type,
ANSI the same collection type,
or the same abstract data type, respectively.

General Rules

1) Let *T* be an object column.

2) If the value of *V* is a null value, then the value of *T* is set to the same null value.

3) Otherwise, let *V* denote a non-null value of *T*.

   Case:

   a) If the data type of *T* is fixed-length character string with length in characters *L* and the length in characters of *V* is equal to *L*, then the value of *T* is set to *V*.

   b) If the data type of *T* is fixed-length character string with length in characters *L* and the length in characters *M* of *V* is larger than *L*, then

      Case:

      1) If the rightmost $M - L$ characters of *V* are all <space>s, then the value of *T* is set to the first *L* characters of *V*.

      2) If one or more of the rightmost $M-L$ characters of *V* are not <space>s, then an exception condition is raised: *data exception—string data, right truncation*.

   c) If the data type of *T* is fixed-length character string with length in characters *L* and the length in characters *M* of *V* is less than *L*, then the first *M* characters of *T* are set to *V* and the last $L-M$ characters of *T* are set to <space>s.

   d) If the data type of *T* is variable-length character string and the length in characters *M* of *V* is not greater than the maximum length in characters of *T*, then the value of *T* is set to *V* and the length in characters of *T* is set to *M*.

   e) If the data type of *T* is variable-length character string and the length in characters *M* of *V* is greater than the maximum length in characters *L* of *T*, then,

Case:

i) If the rightmost $M - L$ characters of V are all <space>s, then the value of T is set to the first L characters of V and the length in characters of T is set to L.

ii) If one or more of the rightmost $M-L$ characters of V are not <space>s, then an exception condition is raised: *data exception—string data, right truncation*.

f) If the data type of T is fixed-length bit string with length in bits L and the length in bits of V is equal to L, then the value of T is set to V.

g) If the data type of T is fixed-length bit string with length in bits L and the length in bits M of V is greater than L, then an exception condition is raised: *data exception—string data, right truncation*.

h) If the data type of T is fixed-length bit string with length in bits L and the length in bits M of V is less than L, then an exception condition is raised: *data exception—string data, length mismatch*.

i) If the data type of T is variable-length bit string and the length in bits M of V is not greater than the maximum length in bits of T, then the value of T is set to V and the length in bits of T is set to M.

j) If the data type of T is variable-length bit string, and the length in bits M of V is greater than the maximum length in bits L of T, then an exception condition is raised: *data exception—string data, right truncation*.

k) If the data type of T is numeric and there is an approximation obtained by rounding or truncation of the numerical value of V for the data type of T, then the value of T is set to such an approximation.

If there is no such approximation, then an exception condition is raised: *data exception—numeric value out of range*.

If the data type of T is exact numeric, then it is implementation-defined whether the approximation is obtained by rounding or by truncation.

l) If the data type of T is datetime and there is a representation of the value of V in the data type of T, then the value of T is set to that representation.

m) If the data type of T is interval and there is a representation of the value of V in the data type of T, then the value of T is set to that representation. Otherwise, an exception condition is raised: *data exception—interval field overflow*.

n) If the data type of T is boolean, then the value of T is set to V.

o) If the data type of T is an enumerated type, then the value of T is set to the value of V.

p) If the data type of T is an abstract data type, then the value of T is set to the value of V.

q) If the data type of T is object identifier, then the value of T is set to V.

ANSI Only–SQL3

r) If the data type of *T* is a collection type, then the value of *T* is set to *V*.

4) Let *V* be a null insert value of *T*.

Case:

a) If the insert value NULL (<null state>) was specified, then *T* is set to the null state specified.

b) If the insert value
ANSI host parameter reference
ISO <parameter specification>
was specified and the
ANSI host parameter reference
ISO <parameter specification>
contains an indicator parameter, then

Case:

i) If the value of the indicator parameter is equal to −1, then *T* is set to the general null value.

ii) If the value of the indicator parameter is equal to the negative of the position of a null state in the null class of *T*, then *T* is set to that null state.

iii) Otherwise, an exception condition is raised: *data exception—invalid indicator parameter value*.

c) Otherwise, *T* is set to the general null value.

5) If the column definition of *T* includes the name of a domain whose domain descriptor includes a domain constraint *D*, then *D* is effectively checked. If *D* is not satisfied, then an exception condition is raised: *integrity constraint violation*.

9.3 Set operation result data types and nullabilities

Function

Specify the Syntax Rules and result data types and nullabilities for
ANSI <case expression>s, <table value expression>s,
ISO <case expression>s
and <query expression>s having set operators.

Syntax Rules

1) Let *DTS* be a set of data types specified in an application of this Subclause.

2) All of the data types in *DTS* shall be comparable.

3) Case:

   a) If any of the data types in *DTS* is character string, then all data types in *DTS* shall be character string, and all of them shall have the same character repertoire. That character repertoire is the character repertoire of the result. The character set of the result is the character set of one of the data types in *DTS*. The specific character set chosen is implementation-dependent. The collating sequence and the coercibility attribute are determined as specified in Table 2, "Collating coercibility rules for dyadic operators".

      Case:

      i) If any of the data types in *DTS* is variable-length character string, then the result data type is variable-length character string with maximum length in characters equal to the maximum of the lengths in characters and maximum lengths in characters of the data types in *DTS*.

      ii) Otherwise, the result data type is fixed-length character string with length in characters equal to the maximum of the lengths in characters of the data types in *DTS*.

   b) If any of the data types in *DTS* is bit string, then all data types in *DTS* shall be bit string.

      Case:

      i) If any of the data types in *DTS* is variable-length bit string, then the result data type is variable-length bit string with maximum length in bits equal to the maximum of the lengths in bits and maximum lengths in bits of the data types in *DTS*.

      ii) Otherwise, the result data type is fixed-length bit string with length in bits equal to the maximum of the lengths in bits of the data types in *DTS*.

   c) If all of the data types in *DTS* are exact numeric, then the result data type is exact numeric with implementation-defined precision and with scale equal to the maximum of the scales of the data types in *DTS*.

   d) If any data type in *DTS* is approximate numeric, then each data type in *DTS* shall be numeric and the result data type is approximate numeric with implementation-defined precision.

   e) If any data type in *DTS* is a datetime data type, then each data type in *DTS* shall be the same datetime data type. The result data type is the same datetime data type.

f) If any data type in *DTS* is interval, then each data type in *DTS* shall be interval. If the precision of any data type in *DTS* specifies YEAR or MONTH, then the precision of each data type shall specify only YEAR or MONTH. If the precision of any data type in *DTS* specifies DAY, HOUR, MINUTE, or SECOND(*N*), then the precision of no data type of *DTS* shall specify the <datetime field>s YEAR and MONTH. The result data type is interval with precision ''*S* TO *E*'', where *S* and *E* are the most significant of the <start field>s and the least significant of the <end field>s of the data types in *DTS*, respectively.

g) If any data type in *DTS* is an enumerated type, then each data type in *DTS* shall be that enumerated type. The result data type is that enumerated type.

h) If any data type in *DTS* is boolean, then each data type in *DTS* shall be boolean. The result data type is boolean.

i) If any data type in *DTS* is an abstract data type, then each data type in *DTS* shall be the same abstract data type.

ANSI Only–SQL3

The result data type is the same abstract data type.

j) If any data type in *DTS* is an object identifier type, then each data type in *DTS* shall be an object identifier type qualified by an abstract data type ADT_i that is in the same abstract data type family. Let *ADT* be the data type that is the most specific single data type that is a supertype of all the ADT_i. The result data type is REF(*ADT*).

k) If any data type in *DTS* is a collection type, then each data type in *DTS* shall be the same collection type.

4) If all of the null classes of the data types in *DTS* that are not the general null class are the same null class, then the result null class is that null class. Otherwise, the result has the general null class.

General Rules

None.

9.4 Subject routine determination

Function

Determine the set of subject routines applicable to a given invocation.

Syntax Rules

1) Let *CF* and *OP* be a set of candidate routines and operands specified in an application of this Subclause.

2) Let CF_i be the *i*-th candidate routine.

3) Let OP_j be the *j*-th <argument> in the <argument list>. Let FP_{ij} be the *j*-th <formal parameter> of CF_i.

4) Type conversions are said to be *preferred* in the following manner:

   a) No conversion (*i.e.*, an exact match) is better than . . .

   b) an implicit conversion to a supertype *S* such that there is no other conversion to a supertype *S2* for which *S* is a supertype of *S2*, which in turn is better than . . .

   c) an implicit SQL-defined CAST, which in turn is better than . . .

   d) an implicit user-defined CAST.

5) For every <argument> in the <argument list> and for each CF_i, determine which conversion, if any, can be used to convert OP_j to the data type of

ANSI Only–SQL3

FP_{ij}, if FP_{ij} is an input parameter, or to convert FP_{ij} to the data type of OP_j if FP_{ij} is an output parameter.

ISO Only—caused by ANSI changes not yet considered by ISO

FP_{ij}.

Select the most preferred conversion such that the conversion is supported for at least one FP_{ij}. Let S_j be the set of all candidate routines CF_i corresponding to the selected FP_{ij}s.

6) The set of subject <routine>s is the intersection of all sets S_j.

10 Additional common elements

10.1 <interval qualifier>

Function

Specify the precision of an interval data type.

Format

```
<interval qualifier> ::=
         <start field> TO <end field>
       | <single datetime field>

<start field> ::=
       <non-second datetime field>
           [ <left paren> <interval leading field precision> <right paren> ]

<end field> ::=
         <non-second datetime field>
       | SECOND [ <left paren> <interval fractional seconds precision> <right paren> ]

<single datetime field> ::=
         <non-second datetime field>
             [ <left paren> <interval leading field precision> <right paren> ]
       | SECOND [ <left paren> <interval leading field precision>
             [ <comma> <interval fractional seconds precision> ] <right paren> ]

<datetime field> ::=
         <non-second datetime field>
       | SECOND

<non-second datetime field> ::= YEAR | MONTH | DAY | HOUR | MINUTE

<interval fractional seconds precision> ::= <unsigned integer>

<interval leading field precision> ::= <unsigned integer>
```

Syntax Rules

1) There is an ordering of significance of <datetime field>s. In order from most significant to least significant, the ordering is: YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. A <start field> or <single datetime field> with an <interval leading field precision> *i* is more significant than a <start field> or <single datetime field> with an <interval leading field precision> *j* if *i*>*j*. An <end field> or <single datetime field> with an <interval fractional seconds precision> *i* is more significant than an <end field> or <single datetime field> with an <interval fractional seconds precision> *j* if *i*>*j*.

2) If TO is specified, then:

   a) <start field> shall be more significant than <end field>,

b) <start field> shall not specify MONTH, and

c) if <start field> specified YEAR, then <end field> shall specify MONTH.

3) The maximum value of <interval leading field precision> is implementation-defined, but shall not be less than 2.

4) The maximum value of <interval fractional seconds precision> is implementation-defined, but shall not be less than 6.

5) An <interval leading field precision>, if specified, shall be greater than 0 and shall not be greater than the implementation-defined maximum. If <interval leading field precision> is not specified, then an <interval leading field precision> of 2 is implicit.

6) An <interval fractional seconds precision>, if specified, shall be greater than or equal to 0 and shall not be greater than the implementation-defined maximum. If SECOND is specified and <interval fractional seconds precision> is not specified, then an <interval fractional seconds precision> of 6 is implicit.

Access Rules

None.

General Rules

1) An item qualified by an <interval qualifier> contains the datetime fields identified by the <interval qualifier>.

Case:

a) If the <interval qualifier> specifies a <single datetime field>, then the <interval qualifier> identifies a single <datetime field>. Any reference to the *most significant* or *least significant* <datetime field> of the item refers to that <datetime field>.

b) Otherwise, the <interval qualifier> identifies those datetime fields from <start field> to <end field>, inclusive.

2) An <interval leading field precision> specifies

Case:

a) If the <datetime field> is SECOND, then the number of decimal digits of precision before the specified or implied decimal point of the seconds <datetime field>.

b) Otherwise, the number of decimal digits of precision of the first <datetime field>.

3) An <interval fractional seconds precision> specifies the number of decimal digits of precision following the specified or implied decimal point in the <datetime field> SECOND.

4) If <single datetime field> is not specified and <start field> and <end field> are the same <datetime field>, then the <interval qualifier> is equivalent to a <single datetime field> that is that <datetime field>.

5) The length in positions of an item of type interval is computed as follows.

Case:

a) If the item is a year-month interval, then

Case:

i) If the <interval qualifier> is a <single datetime field>, then the length in positions of the item is the implicit or explicit <interval leading field precision> of the <single datetime field>.

ii) Otherwise, the length in positions of the item is the implicit or explicit <interval leading field precision> of the <start field> plus 2 (the length of the <non-second datetime field> that is the <end field>) plus 1 (the length of the <minus sign> between the <years value> and the <months value> in a <year-month literal>).

b) Otherwise,

Case:

i) If the <interval qualifier> is a <single datetime field> that does not specify SECOND, then the length in positions of the item is the implicit or explicit <interval leading field precision> of the <single datetime field>.

ii) If the <interval qualifier> is a <single datetime field> that specifies SECOND, then the length in positions of the item is the implicit or explicit <interval leading field precision> of the <single datetime field> plus the implicit or explicit <interval fractional seconds precision>. If <interval fractional seconds precision> is greater than zero, then the length in positions of the item is increased by 1 (the length in positions of the <period> between the <seconds integer value> and the <seconds fraction>).

iii) Otherwise, let *participating datetime fields* mean the datetime fields that are less significant than the <start field> and more significant than the <end field> of the <interval qualifier>. The length in positions of each participating datetime field is 2.

Case:

1) If <end field> is SECOND, then the length in positions of the item is the implicit or explicit <interval leading field precision>, plus 3 times the number of participating datetime fields (each participating datetime field has length 2 positions, plus the <minus sign>s or <colon>s that precede them have length 1 position), plus the implicit or explicit <interval fractional seconds precision>, plus 1 (the length in positions of the <colon> preceding the <end field>). If <interval fractional seconds precision> is greater than zero, then the length in positions of the item is increased by 1 (the length in positions of the <period> within the field identified by the <end field>).

2) Otherwise, the length in positions of the item is the implicit or explicit <interval leading field precision>, plus 3 times the number of participating datetime fields (each participating datetime field has length 2 positions, plus the <minus sign>s or <colon>s that precede them have length 1 position), plus 2 (the length in positions of the <end field>), plus 1 (the length in positions of the <colon> preceding the <end field>).

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <interval qualifier>.

10.2 <language clause>

Function

Specify a standard programming language.

Format

```
<language clause> ::=
       LANGUAGE <language name>

<language name> ::=
       ADA | C | COBOL | FORTRAN | MUMPS | PASCAL | PLI | SQL
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) The standard programming language specified by the clause is defined in the
   ANSI American
   ISO International
   Standard identified by the <language name> keyword. Table 16, "Standard programming languages", specifies the relationship.

Table 16—Standard programming languages

ANSI Only–SQL3

Language keyword	Relevant standard
ADA	ANSI/MIL-STD-1815A
C	ANSI X3.159
COBOL	ANSI X3.23
FORTRAN	ANSI X3.9 and ANSI X3.198
MUMPS	ANSI/MDC X11.1
PASCAL	ANSI/IEEE 770/X3.97 and ANSI/IEEE 770/X3.160
PLI	ANSI X3.53
SQL	ANSI X3.135-1992

ISO Only–SQL3

ADA	ISO/IEC 8652
C	ISO/IEC 9899
COBOL	ISO 1989
FORTRAN	ISO 1539
MUMPS	ISO/IEC 11756
PASCAL	ISO 7185 and ISO/IEC 10206
PLI	ISO 6160
SQL	ISO 9075:1992

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <language clause> shall not specify MUMPS.

10.3 <routine invocation>

Function

Invoke a routine.

Format

```
<routine invocation> ::=
       <routine name> <argument list>

<argument list> ::=
       <left paren> [ <argument> [ { <comma> <argument> }... ] ] <right paren>

<argument> ::=
         <value expression>
       | <generalized expression>

<generalized expression> ::=
       <value expression> AS <abstract data type name>
```

Syntax Rules

1) The <data type> of the <value expression> of a <generalized expression> shall be a subtype of the <data type> denoted by the <abstract data type name> of that <generalized expression>.

2) A <generalized expression> shall be contained in an SQL routine.

3) Case:

   a) If an <argument> is a <generalized expression>, then the *typeset* for the <argument> contains a single member that is the data type denoted by the <abstract data type name> of the <generalized expression>.

   b) If an <argument> is a <value expression> or <status parameter> and the <data type> of the <argument> is not part of a subtype family, or if the <argument> does not correspond with an input parameter, then the *typeset* for the <argument> contains a single member that is the data type of the <argument>.

   c) Otherwise, the *typeset* of the <argument> contains all <data type>s that are subtypes of the data type of the <argument>.

ANSI Only–SQL3

---

4) Let *RN* be the <routine name> of a <routine invocation> *RI*, let *QI* be the <qualified identifier> of *RN*, and let *NA* be the number of <argument>s in the <argument list> of *RI*.

5) A ‘‘possibly candidate routine’’ is a <routine> whose <routine name>’s <qualified identifier> is equal to *RN* and whose <parameter list> contains *NA* <parameter declaration>s.

6) If *RN* does not contain a <local or schema qualifier>, then:

Case:

a) If *RI* is contained in a <compound statement> whose <local declaration list> contains a possibly-candidate routine *R2*, then the candidate <routine>s of *RI* are the routine family of *R2*.

Note: ''routine family'' is defined in Subclause 13.2, "<routine>".

b) If *RI* is contained in a <module> whose <module contents> contains a possibly-candidate routine *R2*, then the candidate <routine>s of *RI* are the routine family of *R2*.

Note: ''routine family'' is defined in Subclause 13.2, "<routine>".

c) Otherwise:

i) If *RI* is contained in a <module>, then let *S* be the schema that is specified or implicit for that <module>. If *RI* is contained in a <schema>, then let *S* be the <schema name> that is specified or implicit in the <schema definition>

ii) *S* shall contain at least one possibly-candidate routine *R2*. The candidate <routine>s of *RI* are the routine family of *R2*.

Note: ''routine family'' is defined in Subclause 13.2, "<routine>".

7) If *RN* contains a <local or schema qualifier> *LSQ*, then

a) If *RI* is contained in a <compound statement> *CS* whose <statement label> is equal to *LSQ*, then the <local declaration list> of *CS* shall contain at least one possibly-candidate routine *R2*. The candidate <routine>s of *RI* are the routine family of *R2*.

Note: ''routine family'' is defined in Subclause 13.2, "<routine>".

b) If *LSQ* is ''MODULE'' or *RI* is contained in a <module> *M* whose <module name> is equal to *LSQ*, then *M* shall contain at least one possibly-candidate routine *R2*. The candidate <routine>s of *RI* are the routine family of *R2*.

Note: ''routine family'' is defined in Subclause 13.2, "<routine>".

c) Otherwise, *LSQ* shall be the <schema name> of a <schema> that contains at at least one possibly-candidate routine, *R2*. The candidate <routine>s of *RI* are the routine family of *R2*.

Note: ''routine family'' is defined in Subclause 13.2, "<routine>".

ISO Only—caused by ANSI changes not yet considered by ISO

8) The <routine name> of a <routine invocation> shall be specified in one or more <routine>s that exist in the same catalog as the <routine invocation> and that have the same number of <parameter declaration>s as the number of <argument>s in the <argument list> of the <routine invocation>. Those <routine>s are the candidate <routine>s of the <routine invocation>.

9) Let *AL* be the <argument list> of a <routine invocation>. Let *XAL* be the set of all <argument list>s XAL_i derived from *AL* by taking the cross product of the <argument>'s typesets. The set *XAL* represents all possible combinations of most-specific data types present in *AL* when the <argument>s are evaluated. For each such XAL_i, the Syntax Rules of Subclause 9.4, "Subject

routine determination", are applied to the candidate <routine>s and operands of XAL_i, yielding a set of subject <routine>s. For each XAL_i, there shall be
ANSI at most
ISO exactly
one subject <routine> in the set.

ANSI Only–SQL3

10) If there is any XAL_i for which the set of subject routines is empty, then:

a) The <argument list> immediately contained in the <routine invocation> shall contain exactly one <argument>, and the data type of that argument shall be a collection type.

b) For each XAL_i for which the set of subject routines is empty, let CT_i be the collection type, and ET_i be the element type of its single argument. The Syntax Rules of this Subclause are applied to a <routine invocation> containing <routine name> *XRN* and a single <argument>, where that <argument> contains an arbitrary <value expression> whose data data type is ET_i. For each such XAL_i, the application of these rules shall yield exactly one subject <routine> that will be referred to as an *iterable subject routine*; let ISR_i be that iterable subject routine. The single parameter of ISR_i shall be an input parameter.

11) For each ISR_i that is a function, let its <returns data type> be RDT_i. The effective <returns data type> of ISR_i is:

Case:

i) If CT_i is a MULTISET or SET collection type, then MULTISET(RDT_i).

ii) If CT_i is LIST, then LIST(RDT_i).

12) All of the identified subject <routine>s shall be functions.

13) For any subject <routine> that is not an iterable subject routine, its *effective returns data type* is its <returns data type>.

14) Case:

i) If there is a single identified subject <routine>, then the data type of the <routine invocation> is the effective returns data type of that <routine>.

ii) Otherwise, let RT_i be the effective returns data type for the identified subject <routine> for each XAL_i. All data types RT_i shall be members of the same subtype family. The data type of the <routine invocation> is the most-specific single data type that is a supertype of all of the RT_i.

15) If any subject <routine> is a function, then:

a) All of the identified subject <routine>s shall be functions.

b) For any subject <routine> that is not an iterable subject routine, its *effective returns data type* is its <returns data type>.

c) Case:

i) If there is a single identified subject <routine>, then the data type of the <routine invocation> is the effective returns data type of that <routine>.

ii) Otherwise, let RT_i be the effective returns data type for the identified subject <routine> for each XAL_i. All data types RT_i shall be members of the same subtype family. The data type of the <routine invocation> is the most-specific single data type that is a supertype of all of the RT_i.

ISO Only—caused by ANSI changes not yet considered by ISO

16) For each subject <routine> F, let A_i be the i-th <argument> in the <argument list> P_i be the i-th parameter of F, and T_i be the <data type> of P_i.

Case:

a) If P_i is an input parameter, then A_i shall be such that

CAST (A_i AS T_i)

is valid according to the Syntax Rules of Subclause 6.13, "<cast specification>".

b) If P_i is an output parameter, then let V_i be an arbitrary value expression of data type T_i and let AT_i be the data type of A_i. The <argument> A_i shall be a <value specification> that is a <target specification>. AT_i shall be such that

CAST (V_i AS AT_i)

is valid according to the Syntax Rules of Subclause 6.13, "<cast specification>".

17) If the subject <routine>s are functions, then

Case:

a) If there is a single identified subject <routine>, then the data type of the <routine invocation> is the <returns data type> of that <routine>.

b) Otherwise, let RT_i be the <returns data type> for the identified subject <routine> for each XAL_i. All data types RT_i shall be members of the same subtype family. The data type of the <routine invocation> is the most-specific single data type that is a supertype of all of the RT_i.

18) If the subject <routine> specifies DESTRUCTOR, then:

a) If the ADT instance to be destroyed is in a column of a row of a table, then the <routine invocation> shall be contained in a <routine> that is a destructor for the abstract data type instance containing that column.

b) If the ADT instance to be destroyed is in a row of a table, then the <routine invocation> shall be implied by the rules of a <delete statement: positioned> or a <delete statement: searched>.

Access Rules

1) For any subject <routine> that is contained in a <module> or that is an external <routine>, the applicable privileges shall include the EXECUTE privilege on the <module>.

General Rules

1) Evaluate, in any order, each <argument> contained in the <argument list> of the <routine invocation> to obtain a value.

ANSI Only–SQL3

2) Let *PL* be the evaluated <argument>s, and let XAL_j be the member of *XAL* whose <argument>s have the same data types as *PL* in the same order. The subject <routine> to be invoked is the one yielded for XAL_j by the Syntax Rules of this Subclause, and it has effective <returns data type> RT_j.

3) If the <routine> is not an iterable subject <routine>, then let *N* be the number of <argument>s of the <argument list>, and for *i* ranging from 1 to *N*, let PV_i be the value of the *i*-th <argument>, T_i be the data type of the *i*-th <parameter declaration> of the <parameter list> of the subject <routine>, and let CPV_i be the result of

ISO Only—caused by ANSI changes not yet considered by ISO

4) Let *PL* be the evaluated <argument>s. The Syntax Rules of Subclause 9.4, "Subject routine determination", are applied to the candidate <routine>s and operands of *PL*, yielding a single subject <routine>.

5) Let *N* be the number of <argument>s of the <argument list>. For *i* ranging from 1 to *N*, let PV_i be the value of the *i*-th <argument>, T_i be the data type of the *i*-th <parameter declaration> of the <parameter list> of the subject <routine>, and let CPV_i be the result of

CAST (PV_i AS T_i)

ANSI Only–SQL3

6) If the <routine> is an SQL routine, then:

Case:

a) If the <routine> is not an iterable subject <routine>, then invoke the subject <routine> and supply the values CPV_i as an argument value list. If the subject <routine> is a function, then let *R* be the returned value.

b) Otherwise,

i) Let *AV* be the value of the single <argument>.

ii) Let *:ARGUMENT* and *:RESULT* be implementation-dependent <variable name>s different from each other and from the <variable name> of any other variable with overlapping scope.

iii) If RT_j is a MULTISET or SET collection type, then let the initial result value *IRV* be MULTISET( ); otherwise, let *IRV* be LIST( ).

iv) Let *CTI* be CT_i, let *ETI* be ET_i, let *RTI* be RT_i, and let *ISRJ* be ISR_j.

v) The <routine invocation> is equivalent to *:RESULT*, where the value of *:RESULT* is effectively computed by the following <compound statement>:

```
DECLARE :ARGUMENT CTJ(ETJ) = AV;
        :RESULT RTJ = IRV;
BEGIN
  FOR X IN TABLE :ARGUMENT DO
    INSERT INTO :RESULT VALUES ISRJ(X);
  END FOR;
END;
```

ISO Only—caused by ANSI changes not yet considered by ISO

7) If the <routine> is an SQL routine, then invoke the subject <routine> and supply the values CPV_i as an argument value list. If the subject <routine> is a function, then let *R* be the returned value.

8) If the subject <routine> is an external routine, then:

a) Let *N* be the number of <argument>s. Supply $N + 1$ indicator arguments whose data type is exact numeric with a scale of 0. Set the value of indicator argument $N + 1$ to 0. If CPV_i is a null value, then set the value of the *i*-th indicator argument to -1; otherwise, set the value of the *i*-th indicator argument to 0.

b) Let *PN* be the number of data types T_i that are predefined types. Let *UN* be the number of base items in the data types of T_i that are user-defined types. If the subject <routine> is a procedure, then let *RN* be 0. If the subject <routine> is a function, then let *RT* be the <returns data type>. If *RT* is a user-defined type, then let *RN* be the number of base items in that data type; otherwise, let *RN* be 1.

ANSI Only–SQL3

c) The exception data item is set to '00000'.

d) Supply an argument value list of $PN + UN + RN + N + 2$ entries:

i) Argument value list entries 1 to $PN + UN$ are *N* groups of entries:

ii) For *i* ranging from 1 to *N*: if T_i is a predefined type, then the *i*-th group of argument value list entries consists of the single value CPV_i; if T_i is a user-defined type, then the *i*-th group of argument value list entries consists of the base items of value CPV_i.

iii) If the subject <routine> is a function, then argument value list entries $(PN + UN) + 1$ to $(PN + UN) + RN$ are as follows:

1) If *RT* is a predefined type, then argument value list entry $(PN + UN) + 1$ is the single result data item.

2) If *RT* is a user-defined type, then argument value list entries $(PN + UN) + 1$ to $(PN + UN) + RN$ are the *RN* base items of the result data item.

iv) Argument value list entries $(PN + UN + RN) + 1$ to $(PN + UN + RN) + N + 1$ are the $N + 1$ indicator arguments.

v) Argument value list entry $(PN + UN + RN + N + 1) + 1$ is the exception
ISO subclass
data item.

e) The value of the implicit or explicit <external routine name> identifies an external routine. The method and time of binding of the designated external routine to the schema or <module> containing the <routine> is implementation-defined.

• 1 sentence deleted.

f) If the subject <routine> specifies NOT VARIANT and if different invocations of the external routine with identical argument value lists do not produce identical results, then the results are implementation-dependent.

g) Invoke the external routine of the subject <routine> with the argument value list.

h) If the <external routine language clause> specifies ADA (respectively C, COBOL, FORTRAN, MUMPS, PASCAL, PLI) and the external routine designated by the <routine name> is not a standard Ada program (respectively C, COBOL, Fortran, MUMPS, Pascal, PL/I program), then the effect of the <routine invocation> is implementation-dependent.

ANSI Only–SQL3

i) After execution of the external routine completes,

Case:

i) If the exception data item is '00000', then the external routine was successful.

ii) If the first two characters of the exception data item are equal to the SQLSTATE exception code class value for *external function exception*, then an exception condition is raised: *external function exception*, using a subclass code equal to the final three characters of the exception data item.

iii) If the first two characters of the exception data item are equal to the SQLSTATE exception code class value for *warning* and the third characater of the exception data item is 'H', then a completion condition is raised: *warning*, using a subclass code equal to the final three characters of the exception data item.

iv) Otherwise, an exception condition is raised: *external function call exception—invalid SQLSTATE returned*.

ISO Only—caused by ANSI changes not yet considered by ISO

j) After execution of the external routine completes, if the value of the condition subclass data item is not '000', then an exception condition is raised: *external function invocation exception*, with the value of the condition subclass data item as the exception condition subclass.

k) If the subject <routine> is a function, then: If the value of indicator argument N+1 is non-negative, then let *R1* be the value of the result data item. Otherwise, let *R1* be the general null value.

Case:

i) If the subject <external function declaration> contains a <result cast>, then let *RT* be the <returns data type> of the subject <routine>. Let *R* be the result of

CAST (*R1* AS *RT*)

ii) Otherwise, let *R* be *R1*.

l) If the *i*-th <parameter declaration> of the subject <routine> is an output parameter, then: If the *i*-th indicator argument value is negative, then the value of the *i*-th argument is the general null value; otherwise, the value of the *i*-th argument is the corresonding group of argument values.

9) For each <parameter declaration> of the subject <routine> that is an output parameter, let *AV* be the corresponding argument value, let *TS* be the <target specification> of the corresponding <argument>, and let *TT* be the <data type> of *TS*. The value of *TS* is set to the result of

CAST (*AV* AS *TT*)

10) If the subject <routine> is a function, then *R* is the result of the <routine invocation>.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) Conforming Full SQL language shall contain no <routine invocation>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

10.4 <generated type reference>

Function

Specify a type generated from a type template.

Format

```
<generated type reference> ::=
       <type template name> <template parameter list>

<template parameter list> ::=
       <left paren>
         <template parameter> [ { <comma> <template parameter> }... ]
       <right paren>

<template parameter> ::=
         <value specification>
       | <data type>
```

Syntax Rules

1) A <template parameter> that is a <value specification> shall be a <literal> or a <template parameter name>.

2) The <type template name> shall be specified in one or more <type template definition>s that exist in the same catalog as the <generated type reference>. Those <type template definition>s are the candidate <type template definition>s of the <generated type reference>. Let *TPL* be the <template parameter list>.

3) The Syntax Rules of Subclause 9.4, "Subject routine determination" (replacing ''function'' with ''type template''), are applied to the candidate <type template definition>s and operands of *TPL* yielding a set of subject <type template definition>s. There shall be exactly one subject <type template definition> in the set.

4) Let *CTT* be the identified candidate <type template definition>. Let *GT* be an <abstract data type definition> derived from *CTT* as follows:

   a) The keyword TEMPLATE is removed.

   b) The <template parameter declaration list> is removed.

   c) Each occurrence of a <template parameter name> in the <abstract data type body> of *CTT* is replaced with the corresponding <template parameter> of the <generated type reference>.

   d) Two <generated type reference>s are said to be matching if

      i) the <type template name> and <template parameter list> of each identifies the same type template family, and

      ii) the number of <template parameter>s of each is the same, and

iii) for each pair of corresponding parameters:

1) they both identify the same <data type>, or

2) the values of each are equal.

e) The <type template name> is replaced by an implementation-dependent name *IDN* such that:

i) a subsequent matching <generated type reference> yields the same implementation-dependent name *IDN*, and

ii) a subsequent non-matching <generated type reference> yields a different implementation-dependent name.

f) Each occurrence of :GEN_TYPE is replaced by *IDN*.

5) *GT* shall conform to the Syntax Rules for <abstract data type definition>.

6) The <generated type reference> is effectively replaced by *IDN*.

Access Rules

1) The applicable privileges shall include USAGE for the type template family identified by <type template name>.

General Rules

1) If an <abstract data type definition> for *IDN* does not exist, then:

a) The General Rules for the statement *GT* are executed.

b) The abstract data type descriptor is augmented to identify the <type template name> of the <generated type reference>.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) Conforming Full SQL language shall contain no <generated type reference>.

2) The following restrictions apply for Intermediate SQL:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

10.5 <privileges>

Function

Specify privileges.

Format

```
<privileges> ::=
         ALL SCHEMA PRIVILEGES
       | <object privileges> ON <object name>

<object name> ::=
```

ANSI Only--SQL3

```
         [ <table type> ] <table name>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
         [ TABLE ] <table name>

       | DOMAIN <domain name>
       | COLLATION <collation name>
       | CHARACTER SET <character set name>
       | TRANSLATION <translation name>
       | NULL CLASS <null class name>
       | DATA TYPE <abstract data type name>
       | MODULE <module name>
       | TYPE TEMPLATE <type template name>
```

ANSI Only--SQL3

```
       | EXTERNAL ROUTINE { SPECIFIC <specific name> | <member name> }
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
       | EXTERNAL ROUTINE <routine name> [ <data type list> ]
```

```
<object privileges> ::=
         ALL PRIVILEGES
       | <action> [ { <comma> <action> }... ]

<action> ::=
         SELECT [ <left paren> <privilege column list> <right paren> ]
       | DELETE
       | INSERT [ <left paren> <privilege column list> <right paren> ]
       | UPDATE [ <left paren> <privilege column list> <right paren> ]
       | REFERENCES [ <left paren> <privilege column list> <right paren> ]
       | USAGE
       | TRIGGER
       | EXECUTE
       | UNDER
```

```
<privilege column list> ::= <column name list>

<grantee> ::=
          PUBLIC
        | <authorization identifier>
        | <role name>
```

Syntax Rules

1) If the <object name> of the <grant statement> or <revoke statement> specifying <privileges> specifies <table name>, then let *T* be the table identified by that <table name>. *T* shall not be a declared local temporary table.

ANSI Only–SQL3

If a <table type> contained in <object name> is not TABLE, then it shall be the same as that in the descriptor for the table identified by the <table name> contained in <object name>.

---

2) If <object name> specifies a <domain name>, <collation name>, <type template name>, <character set name>, <translation name>, or <null class name>, then <privileges> shall specify USAGE. If <object name> specifies an <abstract data type name>, then <privileges> may specify USAGE. Otherwise, USAGE shall not be specified.

3) If <object name> specifies an <abstract data type name> or <table name>, then <privileges> may specify UNDER; otherwise, UNDER shall not be specified.

4) If <object name> specifies EXTERNAL ROUTINE or MODULE, then <privileges> shall specify EXECUTE; otherwise, EXECUTE shall not be specified.

5) If *T* is a temporary table, then <privileges> shall specify ALL PRIVILEGES.

6) The <privileges> specify one or more privileges on the identified <schema element>s as follows:

   a) ALL SCHEMA PRIVILEGES specifies all privileges on all <schema element>s in the containing schema, including <schema element>s subsequently added by Schema Manipulation Language statements.

   b) The <object privileges> specify one or more privileges on the object identified by <object name>.

7) Each <column name> in a <privilege column list> shall identify a column of *T*.

8) UPDATE (<privilege column list>) is equivalent to the specification of UPDATE (<column name>) for each <column name> in <privilege column list>. INSERT (<privilege column list>) is equivalent to the specification of INSERT (<column name>) for each <column name> in <privilege column list>. REFERENCES (<privilege column list>) is equivalent to the specification of REFERENCES (<column name>) for each <column name> in <privilege column list>. SELECT (<privilege column list>) is equivalent to the specification of SELECT (<column name>) for each <column name> in <privilege column list>.

9) ALL PRIVILEGES is equivalent to the specification of all of the privileges on <object name> for which the current <authorization identifier> has grantable privilege descriptors.

10) If EXTERNAL ROUTINE is specified, then:

ANSI Only–SQL3

a) If <specific name> is specified, then:

i) Let *SR* be the routine identified by the <specific name>.

ii) Let *RF* be the routine family identified by the <routine name> of *SR*.

b) If <routine name> is specified, then:

i) Let *RF* be the routine family identified by the <routine name> *SRN*.

ii) *RF* shall identify a routine family in the catalog identified by the explicit or implicit <catalog name>.

iii) If *RF* identifies a routine family with more than one member, then a <data type list> shall be specified.

iv) If no <data type list> is specified, then the single member *SR* of the routine family is identified.

v) If a <data type list> is specified, then the *i*-th <data type> shall be identical to the *i*-th <data type> in the <parameter declaration list> of a function in *RF*. This is the identified function *SR*.

ISO Only—caused by ANSI changes not yet considered by ISO

c) Let *RF* be the routine family identified by the <routine name> *SRN*.

d) *RF* shall identify a routine family in the catalog identified by the explicit or implicit <catalog name>.

e) If *RF* identifies a routine family with more than one member, then a <data type list> shall be specified.

f) If no <data type list> is specified, then the single member *SR* of the routine family is identified.

g) If a <data type list> is specified, then the *i*-th <data type> shall be identical to the *i*-th <data type> in the <parameter declaration list> of a function in *RF*. This is the identified function *SR*.

Access Rules

None.

General Rules

1) A <grantee> of PUBLIC denotes at all times a list of <grantee>s containing all of the <authorization identifier>s in the SQL environment.

2) The set of applicable privileges for an <authorization identifier> includes those privileges defined by privilege descriptors associated with that <authorization identifier> or by privilege descriptors associated with the enabled applicable roles of that <authorization identifier>, together with those defined by privilege descriptors associated with PUBLIC.

3) SELECT (<column name>) specifies the SELECT privilege on the indicated column and implies one or more column privilege descriptors. If the <privilege column list> is omitted, then SELECT specifies the SELECT privilege on all columns of *T* including any columns subsequently added to *T* and implies a table privilege descriptor and one or more column privilege descriptors.

4) UPDATE (<column name>) specifies the UPDATE privilege on the indicated column and implies one or more column privilege descriptors. If the <privilege column list> is omitted, then UPDATE specifies the UPDATE privilege on all columns of *T* including any column subsequently added to *T* and implies a table privilege descriptor and one or more column privilege descriptors.

5) INSERT (<column name>) specifies the INSERT privilege on the indicated column and implies one or more column privilege descriptors. If the <privilege column list> is omitted, then INSERT specifies the INSERT privilege on all columns of *T* including any column subsequently added to *T* and implies a table privilege descriptor and one or more column privilege descriptors.

6) REFERENCES (<column name>) specifies the REFERENCES privilege on the indicated column and implies one or more column privilege descriptors. If the <privilege column list> is omitted, then REFERENCES specifies the REFERENCES privilege on all columns of *T* including any column subsequently added to *T* and implies a table privilege descriptor and one or more column privilege descriptors.

7) A role *A* identified by <role name> is said to contain the set of roles identified by role authorization descriptors as having been granted to *A*, together with all other roles that are contained by roles in the set.

8) The set of applicable roles for an <authorization identifier> *B* includes all roles defined by role authorization descriptors as having been granted to *B* or to PUBLIC, together with all other roles that are contained in roles already in the set.

9) The set of enabled applicable roles for an <authorization identifier> *B* includes the role *R* enabled by the most recent <set role statement> together with all roles contained in *R*.

10) *B* has the WITH ADMIN OPTION on a role if a role authorization descriptor identifies the role as granted to *B* with the WITH ADMIN OPTION or a role authorization descriptor identifies it as granted with the WITH ADMIN OPTION to another applicable role for *B*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <action> shall not specify EXECUTE, TRIGGER, or UNDER.

   b) An <action> that specifies SELECT shall not contain a <privilege column list>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) An <action> that specifies INSERT shall not contain a <privilege column list>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

10.6 <character set specification>

Function

Identify a character set.

Format

```
<character set specification> ::=
         <standard character repertoire name>
       | <implementation-defined character repertoire name>
       | <user-defined character repertoire name>
       | <standard universal character form-of-use name>
       | <implementation-defined universal character form-of-use name>

<standard character repertoire name> ::= <character set name>

<implementation-defined character repertoire name> ::= <character set name>

<user-defined character repertoire name> ::= <character set name>

<standard universal character form-of-use name> ::=
       <character set name>

<implementation-defined universal character form-of-use name> ::=
       <character set name>
```

Syntax Rules

1) The <standard character repertoire name>s, <implementation-defined character repertoire name>s, <standard universal character form-of-use name>s, and <implementation-defined universal character form-of-use name>s that are supported are implementation-defined.

2) A character set identified by a <standard character repertoire name>, by an <implementation-defined character repertoire name>, by a <standard universal character form-of-use name>, or by an <implementation-defined universal character form-of-use name> has associated with it a privilege descriptor that was effectively defined by the <grant statement>

   GRANT USAGE ON CHARACTER SET *CS* TO PUBLIC WITH GRANT OPTION

   where *CS* is the <character set name> contained in the <character set specification>. The grantor of the privilege descriptor is set to the special grantor value ''_SYSTEM''.

3) The <implementation-defined character repertoire name>s shall include SQL_TEXT.

Access Rules

1) Let *C* be the <character set name> contained in the <character set specification>. The applicable privileges shall include USAGE on *C*.

General Rules

1) A <character set specification> identifies a character set. Let the identified character set be *CS*.

   Note: A character set comprises the characters in the character set's repertoire together with a form-of-use that specifies the convention for arranging those characters into character strings.

2) A <standard character repertoire name> specifies the name of a character repertoire that is defined by a national or international standard. The character repertoire and form-of-use of *CS*, implied by the <standard character repertoire name>, are defined by the standard that defined that <standard character repertoire name>. The default collating sequence of the character repertoire is defined by the order of the characters in the standard and has the PAD SPACE attribute.

3) An <implementation-defined character repertoire name> specifies the name of a character repertoire that is implementation-defined. The character repertoire and form-of-use of *CS*, implied by the <implementation-defined character repertoire name>, are implementation-defined. The default collating sequence of the character repertoire and whether the collating sequence has the NO PAD attribute or the PAD SPACE attribute implementation-defined.

4) A <user-defined character repertoire name> identifies a character set whose descriptor is in some schema whose <schema name> is not INFORMATION_SCHEMA.

   Note: The default collating sequence of the character repertoire is defined as in Subclause 11.33, "<character set definition>".

5) A <standard universal character form-of-use name> identifies form-of-use that is defined by some national or international standard. That form-of-use is the form-of-use of *CS*. The character repertoire of *CS* is as defined in that standard. The default collating sequence of the character repertoire is defined by the order of the characters in ISO/IEC DIS 10646 and has the PAD SPACE attribute.

   Note: Specific forms-of-use implied by this rule include ISO 2022 code extension techniques.

6) An <implementation-defined universal character form-of-use name> identifies an implementation-defined form-of-use that shall be the form-of-use of *CS*. The implied character repertoire and default collating sequence of *CS* and whether the collating sequence has the NO PAD attribute or the PAD SPACE attribute are implementation-defined.

   Note: Specific forms-of-use implied by this rule include implementation-defined techniques such as mixed one-octet/two-octet Latin/Kanji or Compound String.

7) There is a character set descriptor for every character set that can be specified by a <character set specification>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <character set specification>.

10.7 <collate clause>

Function

Specify a default collating sequence.

Format

```
<collate clause> ::= COLLATE <collation name>
```

Syntax Rules

None.

Access Rules

1) Let *C* be the <collation name> contained in the <collate clause>. The applicable privileges shall include USAGE on *C*.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <collate clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

10.8 <constraint name definition> and <constraint attributes>

Function

Specify the name of a constraint and its attributes.

Format

```
<constraint name definition> ::= CONSTRAINT <constraint name>

<constraint attributes> ::=
        <constraint check time> [ [ NOT ] DEFERRABLE ]
      | [ NOT ] DEFERRABLE [ <constraint check time> ]

<constraint check time> ::=   INITIALLY DEFERRED | INITIALLY IMMEDIATE
```

Syntax Rules

1) If a <constraint name definition> is contained in a <schema definition>, and if the <constraint name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

2) The <qualified identifier> of <constraint name> shall be different from the <qualified identifier> of the <constraint name> of any other constraint defined in the same schema.

3) If <constraint check time> is not specified, then INITIALLY IMMEDIATE is implicit.

4) Case:

   a) If INITIALLY DEFERRED is specified, then:

      i) NOT DEFERRABLE shall not be specified.

      ii) If DEFERRABLE is not specified, then DEFERRABLE is implicit.

   b) If INITIALLY IMMEDIATE is specified or implicit and neither DEFERRABLE nor NOT DEFERRABLE is specified, then NOT DEFERRABLE is implicit.

Access Rules

None.

General Rules

1) If NOT DEFERRABLE is specified, then the constraint is not deferrable; otherwise it is deferrable.

2) If <constraint check time> is INITIALLY DEFERRED, then the initial constraint mode for the constraint is *deferred*; otherwise, the initial constraint mode for the constraint is *immediate*.

3) If, on completion of any SQL-statement, the constraint mode of any constraint is immediate, then that constraint is effectively checked.

   Note: This includes the cases where <SQL procedure statement> is a <set constraints mode statement>, a <commit statement>, or the statement that causes a constraint with a constraint mode of *initially immediate* to be created.

4) When a constraint is effectively checked, if the constraint is not satisfied, then an exception condition is raised: *integrity constraint violation*. If this exception condition is raised as a result of executing a <commit statement>, then SQLSTATE is not set to *integrity constraint violation*, but is set to *transaction rollback—integrity constraint violation* (see the General Rules of Subclause 16.6, "<commit statement>").

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no explicit <constraint attributes>.

   Note: This means that INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall contain no <constraint name definition>.

11 Schema definition and manipulation

11.1 <schema definition>

Function

Define a schema.

Format

```
<schema definition> ::=
       CREATE SCHEMA <schema name clause>
         [ <schema character set specification> ]
         [ <schema element>... ]



<schema name clause> ::=
         <schema name>
       | AUTHORIZATION <schema authorization identifier>
       | <schema name> AUTHORIZATION <schema authorization identifier>

<schema authorization identifier> ::=
       <authorization identifier>

<schema character set specification> ::=
       DEFAULT CHARACTER SET <character set specification>

<schema element> ::=
         <table definition>
       | <view definition>
       | <domain definition>
       | <null class definition>
       | <character set definition>
       | <collation definition>
       | <translation definition>
       | <assertion definition>
       | <trigger definition>
       | <routine>
       | <abstract data type definition> | <type template definition>
       | <module definition>
       | <grant statement>
       | <role definition>
       | <grant role statement>
```

Syntax Rules

1) If <schema name> is not specified, then a <schema name> equal to <schema authorization identifier> is implicit.

2) If AUTHORIZATION <schema authorization identifier> is not specified, then

Case:

a) If the <schema definition> is contained in a <module> that has a <module authorization identifier> specified, then an <authorization identifier> equal to that <module authorization identifier> is implicit for the <schema definition>.

b) Otherwise, an <authorization identifier> equal to the SQL-session <authorization identifier> is implicit.

3) The <unqualified schema name> of the explicit or implicit <schema name> shall be different from the <unqualified schema name> of the <schema name> of any other schema in the catalog identified by the <catalog name> of <schema name>.

4) If a <schema definition> appears in a <routine> in a <module>, then the effective <schema authorization identifier> and <schema name> during processing of the <schema definition> is the <schema authorization identifier> and <schema name> specified or implicit in the <schema definition>. Other SQL-statements executed in <routine>s in the <module> have the <module authorization identifier> and <schema name> specified or implicit for the <module>.

5) If <schema character set specification> is not specified, then a <schema character set specification> containing an implementation-defined <character set specification> is implicit.

Access Rules

1) The privileges necessary to execute the <schema definition> are implementation-defined.

General Rules

1) A schema *S* is created with a name equal to the explicit or implicit <schema name> and a default character set name equal to the <character set specification> of the explicit or implicit <default character set specification>.

2) The <schema authorization identifier> is the current <authorization identifier> for privilege determination for *S*.

3) Those objects defined by <schema element>s (base tables, views, constraints, domains, assertions, character sets, translations, collations, privileges, abstract data types) and their associated descriptors are effectively created.

4) The explicit or implicit <character set specification> is used as the default character set used for all <column definition>s and <domain definition>s that do not specify an explicit character set.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) Conforming Full SQL language shall not contain any <null class definition>.

b) Conforming Full SQL language shall not contain any <trigger definition>.

c) Conforming Full SQL language shall not contain any <external function declaration>.

d) Conforming Full SQL language shall not contain any <routine>.

e) Conforming Full SQL language shall not contain any <abstract data type definition>.

f) Conforming Full SQL language shall not contain any <type template definition>.

g) Conforming Full SQL language shall not contain any <module definition>.

h) Conforming Full SQL language shall not contain any <role definition>.

i) Conforming Full SQL language shall not contain any <grant role statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <assertion definition>.

b) Conforming Intermediate SQL language shall not contain any <collation definition>.

c) Conforming Intermediate SQL language shall not contain any <translation definition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any <domain definition>.

b) A <schema name clause> shall specify AUTHORIZATION and shall not specify a <schema name>.

c) A <schema character set specification> shall not be specified.

d) Conforming Entry SQL language shall not contain any <character set definition>.

11.2 <drop schema statement>

Function

Destroy a schema.

Format

```
<drop schema statement> ::=
       DROP SCHEMA <schema name> <drop behavior>

<drop behavior> ::= CASCADE | RESTRICT
```

Syntax Rules

1) Let *S* be the schema identified by <schema name>.

2) *S* shall identify a schema in the catalog identified by the explicit or implicit <catalog name>.

3) If RESTRICT is specified, then *S* shall not contain any persistent base tables, global temporary tables, created local temporary tables, views, domains, assertions, external functions, SQL functions, triggers, modules, abstract data types, type template definitions, character sets, collations, or translations.

   Note: If CASCADE is specified, then such objects will be dropped by the effective execution of the SQL schema manipulation statements specified in the General Rules of this Subclause.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name>.

General Rules

1) Let *T* be the <table name> of any base table or temporary table contained in *S*. The following <drop table statement> is effectively executed:

   DROP TABLE *T* CASCADE

2) Let *V* be the <table name> of any view or temporary view contained in *S*. The following <drop view statement> is effectively executed:

   DROP VIEW *V* CASCADE

3) Let *D* be the <domain name> of any domain contained in *S*. The following <drop domain statement> is effectively executed:

   DROP DOMAIN *D* CASCADE

4) Let *A* be the <constraint name> of any assertion contained in *S*. The following <drop assertion statement> is effectively executed:

   DROP ASSERTION *A*

5) Let *CD* be the <collation name> of any collating sequence contained in *S*. The following <drop collation statement> is effectively executed:

DROP COLLATION *CD* CASCADE

6) Let *TD* be the <translation name> of any translation contained in *S*. The following <drop translation statement> is effectively executed:

DROP TRANSLATION *TD*

7) Let *RD* be the <character set name> of any character set contained in *S*. The following <drop character set statement> is effectively executed:

DROP CHARACTER SET *RD*

8) Let *EF* be the <external function name> of any external function contained in *S*. The following <drop external function statement> is effectively executed:

DROP EXTERNAL FUNCTION *EF* CASCADE

9) Let *SF* be the <SQL function name> of any SQL function contained in *S*. The following <drop routine statement> is effectively executed:

DROP SQL FUNCTION *SF* CASCADE

****Editor's Note****

The preceding statement does not appear to be a "<drop routine statement>" as advertised. Have I missed something?

10) Let *ADT* be the <abstract data type name> of any abstract data type contained in *S*. The following <drop abstract data type statement> is effectively executed:

DROP DATA TYPE *ADT* CASCADE

11) Let *TDT* be the <type template name> of any type template contained in *S*. The following <drop type template statement> is effectively executed:

DROP TYPE TEMPLATE *TDT* CASCADE

12) Let *TT* be the <trigger name> of any trigger contained in *S*. The following <drop trigger statement> is effectively executed:

DROP TRIGGER *TT*

13) Let *N* be the <null class name> of any null class contained in *S*. The following <drop null class statement> is effectively executed:

DROP NULL CLASS *N*

14) Let *M* be the <module name> of any module contained in *S*. The following <drop module statement> is effectively executed:

DROP MODULE *M* CASCADE

15) The identified schema and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop schema statement>.

11.3 <table definition>

Function

Define a persistent base table, a created local temporary table, or a global temporary table.

ANSI Only–SQL3

A table may also be given the additional properties of a set or a list.

Format

```
<table definition> ::=
```

ANSI Only--SQL3

```
       CREATE [ <table scope> ] <table type> <table name>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
       CREATE [ <table type> ] TABLE <table name>

         { <table of new type> | <table of existing type> }
         [ ON COMMIT <table commit action> ROWS ]
```

ANSI Only--SQL3

```
<table scope> ::=
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
<table type> ::=

       <global or local> TEMPORARY

<global or local> ::=
         GLOBAL
       | LOCAL
```

ANSI Only--SQL3

```
<table type> ::=
         TABLE
       | SET
       | LIST
```

```
<table commit action> ::=
        PRESERVE
      | DELETE

<table of new type> ::=
        [ <new type name clause> ] [ <oid options> ]
            [ <constant or updatable> ] <table element list>
      | <subtable of new type>

<constant or updatable> ::=
        CONSTANT
      | UPDATABLE

<subtable of new type> ::=
        [ <new type name clause> ] [ <oid options> ] <subtable clause>
            [ [ <constant or updatable> ] <table element list> ]

<table of existing type> ::=
```

ANSI Only--SQL3

```
        OF <existing type name> [ <table element list> ]
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
        OF <sbstract data type name> [ <table element list> ]

      | <subtable of existing type>
```

ANSI Only--SQL3

```
<existing type name> ::=
        <abstract data type name>
      | <predefined type>

<subtable of existing type> ::=
        OF <abstract data type name> <subtable clause> [ <table element list> ]

<new type name clause> ::=
      OF NEW TYPE <abstract data type name>

<oid options> ::=
        WITH OID [ [ NOT ] VISIBLE ]
      | WITHOUT OID

<subtable clause> ::=
      UNDER <supertable clause> [ { , <supertable clause> }... ]

<supertable clause> ::=
      <supertable name>
        [ WITH ( <member renaming element> [ { , <member renaming element> }... ] ) ]

<member renaming element> ::=
      <supertable member name> AS <subtable member name>
```

```
<supertable member name> ::=
         <column name>
       | <routine name>

<subtable member name> ::=
         <column name>
       | <routine name>

<supertable name> ::= <table name>

<table element list> ::=
       <left paren> <table element> [ { <comma> <table element> }... ] <right paren>

<table element> ::=
         <column definition>
       | <routine declaration>
       | <equals clause>
       | <less-than clause>
       | <cast clause>
       | <table constraint definition>
       | <like clause>

<like clause> ::= LIKE <table name>
```

Syntax Rules

ANSI Only–SQL3

1) If a <table definition> is contained in a <schema definition> *SD* and the <table name> contains a <local or schema qualifier>, then that <local or schema qualifier> shall be the same as the implicit or explicit <schema name> of *SD*.

ISO Only—caused by ANSI changes not yet considered by ISO

2) If a <table definition> is contained in a <schema definition>, and if the <table name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

3) The schema identified by the explicit or implicit schema name of the <table name> shall not include a table descriptor whose table name is <table name>.

4) If the <table definition> appears in a <schema definition>, then let *A* be the explicit or implicit <authorization identifier> of the <schema definition>. Otherwise, let *A* be the <authorization identifier> of the current SQL-session.

ANSI Only–SQL3

5) If a <table type> of SET is specified, then the following <table constraint definition> is implicit as a <table element> in the <table element list>:

UNIQUE (*C1*, . . . , *CN*)

where *C1*, . . . , *CN* is the list of all <column name>s in <column definition>s in the order in which they are specified in the <table element list>.

ISO Only—caused by ANSI changes not yet considered by ISO

6) If <new type name clause> is not specified, then ''OF NEW TYPE *ADTN*'' is implicit, where *ADTN* is an implementation-dependent <abstract data type name> different from the <abstract data type name> of any other abstract data type in the schema. Otherwise, let *ADTN* be the <abstract data type name> specified in a <new type name clause> or in ''OF <abstract data type name>''.

7) If LIKE is specified, then:

a) Let *T1* be the table identified in the <like clause>.

b) The <like clause> is effectively replaced by *columns* where *columns* contains a <column definition> for every column that is defined in *T1*, in the order in which the columns appear in *T1*.

Note: <column constraint>s are not included in *columns*; <column constraint>s are effectively transformed to <table constraints>s and are thereby excluded.

ANSI Only–SQL3

8) If <table of new type> is specified and does not contain a <new type name clause>, then ''OF NEW TYPE *ADTN*'' is implicit, where *ADTN* is an implementation-dependent <abstract data type name> different from the <abstract data type name> of any other abstract data type in the schema.

9) *RT*, the *row type* of the table defined by the <table definition>, is specified as follows:

Case:

a) If <table of new type> is specified and its <new type name clause> contains an <abstract data type name> *ADTN*, then let *RT* be *ADTN*.

b) If <table of existing type> is specified and contains ''OF <abstract data type name>'', then let *RT* be that <abstract data type name>.

c) If <table of existing type> is specified and contains ''OF <predefined type>'', then let *RT* be that <predefined type>.

10) If ''OF <abstract data type name>'' is specified, then let *ADT* be the type identified by <abstract data type name>. If *ADT* is a maximal supertype, then there shall not be a <subtable clause> specified; otherwise, for each direct supertype of *ADT*, the <subtable clause> shall contain exactly one <table name> for a table of that type and no <table name> shall refer to a table of any other type.

11) If ''OF <abstract data type name>'' is specified, then:

a) Let the <abstract data type name> identify an abstract data type *ADT*. Let the <table element list>, if specified, be *TEL1*.

b) Each <table element> in *TEL1* shall be a <table constraint definition>.

c) ''OF <abstract data type name>'' is effectively replaced by a <table element list> *TEL2* whose <table element>s in order are formed from the corresponding <member>s of *ADT* by renaming the <member> node as <table element> and retaining all contained components. If *TEL1* exists, then *TEL2* is replaced by a <table element list> whose <table element>s are the <table element>s of *TEL2*, in the order in which they appear in *TEL2*, followed by the <table element>s of *TEL1*, in the order in which they eppear in *TEL1*.

d) If *TEL2* contains a <column definition> containing <column name> OID, then that <column definition> implicitly contains NOT NULL, and also PRIMARY KEY if *TEL2* does not contain PRIMARY KEY already.

Otherwise, let *TEL2* be the <table element list> of the <table definition>.

12) If

ANSI Only–SQL3

''OF <predefined type>'' is specified, then a <column definition> specifying ''CONTENTS *RT*'' is implicit.

13) If the <table definition> contains <subtable of new type>, then let *SC* be the <subtable clause>. *SC* shall not contain the <table name> of any table whose row type is a <predefined type>.

14) Case:

ISO Only—caused by ANSI changes not yet considered by ISO

the <table definition> contains <subtable of new type>, then let *SC* be the <subtable clause>.

Case:

a) If *SC* contains any <table name> for which the associated abstract data type descriptor specifies WITH OID VISIBLE, then:

i) If <oid options> is specified, then it shall be WITH OID VISIBLE.

ii) Otherwise, WITH OID VISIBLE is implicit.

b) If for every <table name> in *SC* the associated abstract data type descriptor specifies WITHOUT OID and <oid options> is not specified, then WITHOUT OID is implicit.

15) If <oid options> is not specified, then WITH OID NOT VISIBLE is implicit.

16) Case:

a) If <table of new type> or <subtable of new type> immediately contains CONSTANT, then CONSTANT is implicit for each <stored column> contained in the <table element list>.

b) If <table of new type> or <subtable of new type> immediately contains UPDATABLE, then there shall exist at least one <stored column> contained in the <table element list> that does not specify CONSTANT.

17) For every <column name> *CN* in a <column definition> whose <data type> *DT* contains ''REF (<abstract data type name>)'', there shall be a <table constraint definition> in *TEL2* that contains:

a) <referencing columns> that contains a <column name list> containing only *CN*, and

b) <referenced table and columns> that contains the <table name> of a table whose type is that named in *DT*.

18) If ON COMMIT is specified, then TEMPORARY shall be specified.

19) If TEMPORARY is specified and ON COMMIT is not specified, then ON COMMIT DELETE ROWS is implicit.

20) A <table of new type> that is not a <subtable of new type> shall contain at least one <column definition>.

21) No <column name> immediately contained in a <stored column> or a <virtual column> contained in a <column definition> of the <table definition>, or contained in a <member renaming element>, shall specify OID.

22) Let *TN* be the <table name>. If WITH OID is specified, then the following <table element> is implicit as the first <table element> in the <table element list>: a <column definition> corresponding to ''OID REFERENCES *TN* NOT NULL UNIQUE''. If there is no PRIMARY KEY specification in the table definition, then a <unique specification> PRIMARY KEY is also defined on this column OID.

****Editor's Note****

The author of X3H2-92-062rev1/OTT-008 adjusted the position of a number of Syntax Rules, both in this Subclause and in other Subclauses, saying that the moved Rule should precede or follow other Syntax Rules that have "implicit" actions. This appears to be falacious based on our statement in Concepts that the Syntax and Access Rules are effectively applied simultaneously! Is there a problem here?

23) The scope of the <table name> is the <table definition>.

24) Case:

a) If a <table definition> *TD* does not specify a <subtable clause>, then the descriptor of the table *T* defined by *TD* includes the name <table name>,
ANSI the <table type>, the row type of the table,
the column descriptor specified by each <column definition> in *TD*,

ANSI Only–SQL3

the constraint descriptor specified by each <table constraint definition> and <column constraint definition> in *TD*, and the routine descriptor specified by each <routine declaration>

ISO Only—caused by ANSI changes not yet considered by ISO

and the function descriptor specified by each <function declaration>

in *TD*. The *i*-th column descriptor in the descriptor of *T* is given by the *i*-th <column definition> of *TD*.

b) If a <table definition> *TD* specifies a <subtable clause>, then the descriptor of the table *T* defined by *TD* includes the name <table name>,
ANSI the <generated type reference> SQL_TABLE(*RT*), where *RT* is the row type of the table,
the column descriptor specified by each <column definition> in *TD*,

ANSI Only–SQL3

the constraint descriptor specified by each <table constraint definition> and <column constraint definition> in *TD*, and the routine descriptor specified by each <routine declaration>

ISO Only—caused by ANSI changes not yet considered by ISO

the function descriptor specified by each <function declaration>

in *TD*, the column descriptor of each column with <encapsulation level> PUBLIC or PROTECTED specified by the <table definition> TD_i of each <supertable name> T_i specified in the <subtable clause> of *TD*, and the function descriptor of each function with <encapsulation level> PUBLIC or PROTECTED specified by the <table definition> TD_i of each <supertable name> T_i specified in the <subtable clause> of *TD*.

Let *n* be the sum of the number of distinct inherited columns from all direct supertables in the <subtable clause> of *TD*, excluding any column whose <column name> is OID (these columns are not inherited). Let *m* be the number of <column definition>s specified in *TD*. For every *i* less than or equal to *n*, the *i*-th column descriptor in the descriptor of *T*, CD_i, is given by the <column definition> of some column in one of the supertables; the column described by CD_i is called an *inherited column*. For every *j* greater than *n* and less than or equal to $n + m$, the *j*-th column descriptor in the descriptor of *T*, CD_j, is given by the $j - n$-th <column definition> of *TD*; the column described by CD_j is called an *originally defined column*.

The order of the inherited columns in the subtable is determined as follows:

i) Order all the direct supertables of *T* in the order of the <supertable name>s appearing in the <subtable clause> of *TD*.

ii) Construct the list of columns whose descriptors appear in the descriptor of *T* as follows:

$$C_{11}, \cdots, C_{1k(1)}, \cdots, C_{i1}, \cdots, C_{ik(i)}, \cdots, C_{m1}, \cdots, C_{mk(m)}$$

where *m* is the number of direct supertables, *k(i)* is the degree of the *i*-th supertable T_i, C_{ij} is the *j*-th column in T_i, and $C_{i1}, \cdots, C_{ik(i)}$ preserves the order of columns in T_i.

iii) Let *the same column* mean a column appearing in more than one direct supertable of *T* that is inherited by each of those direct supertables from the same column of a single higher-level supertable. The <column name> of the same column in a direct supertable of *T* need not be the same as in other direct supertables of *T*.

iv) If some column *SC* is the same column in more than one direct supertable of *T*, then all occurrences of *SC* except the first occurrence are dropped from the list of columns.

v) For every <member renaming element>, the <column name> of the inherited column identified by the <supertable member name> is replaced by the <subtable member name>.

25) If a <table definition> *TD* specifies a <subtable clause>, then:

a) The table *T* defined by *TD* shall not have *T* as a supertable.

b) No <supertable name> shall appear more than once in the <subtable clause> of *TD*.

c) The <table definition> of any table identified by a <supertable name> in the <subtable clause> of *TD* shall contain an explicit or implicit <unique specification> that specifies PRIMARY KEY.

d) Each <supertable member name> in a <supertable clause> shall identify a column or function, with <encapsulation level> PUBLIC or PROTECTED, of the supertable identified by the <supertable name> in that <supertable clause>, and the same column or function shall not be identified more than once.

e) There shall be exactly one maximal supertable in the subtable family to which the supertable belongs.

26) If ''OF NEW TYPE <abstract data type name>'' is specified, then let *NADTN* be that <abstract data type name>. The abstract data type descriptor for *NADTN* contains the type name *NADTN*, the <oid options> specified, descriptors for all members of *TEL2* with the exclusion of any <unique specification> or <references specification>, and supertype names that are the names of the types of the tables corresponding to each <table name> in the <subtable clause>.

Access Rules

1) If a <table definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <table name>.

ANSI Only–SQL3

2) If a <like clause> is contained in a <table definition>, then *A* shall have SELECT privilege on the table identified in that <like clause>.

ISO Only—caused by ANSI changes not yet considered by ISO

3) *A* shall have SELECT privilege on *T1*.

4) If UNDER is specified,) then *A* shall have UNDER privilege on each table identified by a <supertable clause> in the <subtable clause> and

   Case:

   a) If OF <abstract data type name> is specified, then USAGE privilege on the identified ADT;

   b) Otherwise, UNDER privilege on the types corresponding to each table in the <subtable clause>.

General Rules

1) A <table definition> defines either a persistent base table, a global temporary table or a created local temporary table. If GLOBAL is specified, then a global temporary table is defined. If LOCAL is specified, then a created local temporary table is defined. Otherwise, a persistent base table is defined.

2) The degree of the table being created is initially set to 0; the General Rules of Subclause 11.4, "<column definition>" specify the degree of the table being created during the definition of columns in that table.

3) A table descriptor is created that describes the table being defined.

   a) The name included in the table descriptor is <table name>.

   ANSI Only–SQL3

   b) The table descriptor includes the specified <table type>.

   c) The table descriptor includes the degree of the table, which is the number of <table element>s in the <table definition> that are <column definition>s.

   ANSI Only–SQL3

   d) The table descriptor contains a <generated type reference> SQL_TABLE(*RT*), where *RT* is the row type of the table.

****Editor's Note****

The absence of GRs that explicitly say that the table is created when the <table definition> is executed as an <SQL procedure statement> in a <routine> has been identified as a Possible Problem. This applies to other Subclauses as well. See Possible Problem **205** in the Editor's Notes.

4) A set of privilege descriptors is created that define the privileges INSERT, SELECT, UPDATE, DELETE, and REFERENCES on this table and SELECT, INSERT, UPDATE, and REFERENCES for every <column definition> in the table definition to the <authorization identifier> of the <schema definition> or <module> in which the <table definition> appears. These privileges are grantable.

   The grantor for each of these privilege descriptors is set to the special grantor value ''_SYSTEM''.

5) If a <table definition> *TD* specifies a <subtable clause>, then:

   a) The <table name> *T* of *TD* defines a subtable, called a *direct subtable*. Each <supertable name> specified in the <subtable clause> defines a supertable, called a *direct supertable*.

   b) Every <table element> of every supertable is inherited by the subtable *T*.

ANSI Only–SQL3

6) Case:

   a) If either ''OF <abstract data type name>'' or ''OF NEW TYPE <abstract data type name>'' is specified, then the type of the objects in the table is <abstract data type name>.

   b) Otherwise, the type of the objects in the table is implementation-dependent.

7) The table being created is a multiset table.

Leveling Rules

1) The following restrictions apply for Full SQL:

ANSI Only–SQL3

   a) A <table definition> shall not specify a <constant or updatable>.

   b) A <table definition> shall not specify a <table of existing type>.

   c) A <table definition> shall not specify a <new type name clause>.

   d) A <table definition> shall not specify a <subtable clause>.

   e) A <table element> shall not be a <routine declaration>. A <table definition> shall not contain <oid options>.

ISO Only—caused by ANSI changes not yet considered by ISO

   f) A <table source> shall not be a <new object table definition>.

   g) A <table source> shall not be an <existing object table definition>.

h) A <table source> shall not be a <subtable definition>.

i) A <base table definition> shall not contain a <new type specification>.

j) A <table element> shall not be a <function declaration>.

---

k) A <table element> shall not be an <equals clause>.

l) A <table element> shall not be a <less-than clause>.

m) A <table element> shall not be a <like clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not specify TEMPORARY and shall not reference any global or local temporary table.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.4 <column definition>

Function

Define a column of a table.

Format

```
<column definition> ::=
         <stored column>
       | <virtual column>

<stored column> ::=
       [ <encapsulation level> ]
         <column name> [ <constant or updatable> ]
           [ <data type> | <domain name> ]
       [ <default clause> ]
       [ <column constraint definition>... ]
       [ <collate clause> ]
       [ <null clause> ]

<virtual column> ::=
       [ <encapsulation level> ]
         <column name> <derivation clause>
       [ <check constraint definition>... ]
       [ <collate clause> ]

<derivation clause> ::=
```

ANSI Only--SQL3

```
       [ READ ONLY | CONSTANT | UPDATABLE ] [ <data type> ]
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
       [ READ ONLY | CONSTANT | UPDATABLE ] <data type>

         VIRTUAL [ <derivation functions> ]

<derivation functions> ::=
         <get function> [ <set function> ]
       | <set function> [ <get function> ]

<get function> ::=
       GET WITH <routine name>

<set function> ::=
       SET WITH <routine name>

<encapsulation level> ::=
         PRIVATE
       | PROTECTED
       | PUBLIC

<column constraint definition> ::=
       [ <constraint name definition> ]
       <column constraint> [ <constraint attributes> ]
```

```
<column constraint> ::=
         NOT NULL
       | <unique specification>
       | <references specification>
       | <check constraint definition>

<null clause> ::=
       NULL IS <null class name>
```

Syntax Rules

1) Let *TD* be the <table definition> or <abstract data type definition> that contains this <column definition>. If this is the first <column definition> or <routine declaration> in *TD* and no <encapsulation level> is specified, then PUBLIC is implicit; otherwise if no <encapsulation level> is specified, then the <encapsulation level> of the immediately preceding <column definition> or <routine definition> in *TD* is implicit.

2) Case:

   a) If the <column definition> is contained in a <table definition>, then let *T* be the table defined by that <table definition>.

   b) If the <column definition> is contained in a <temporary table declaration>, then let *T* be the table declared by that <temporary table declaration>.

   c) If the <column definition> is contained in an <alter table statement>, then let *T* be the table identified in the containing <alter table statement>.

   d) If the <column definition> is contained in an <abstract data type definition>, then let *T* be the abstract data type identified in that <abstract data type definition>.

   The <column name> in the <column definition> shall be different from the <column name> of any column of *T*.

3) Case:

   a) If a <column definition> is contained in a <table definition> or <abstract data type definition> *TD* that does not specify a <subtable clause>, then the *i*-th column of the table or abstract data type defined by *TD* is described by the *i*-th <column definition> in *TD*. The encapsulation level, name, and the data type or domain of the column are specified by the <encapsulation level> <column name> and <data type> or <domain name>, respectively.

   b) If a <column definition> is contained in a <table definition> *TD* that specifies a <subtable clause>, then let *n* be the sum of the number of distinct inherited columns of all <supertable name>s specified in the <subtable clause> of *TD*. The *i* +*n*-th column of the table defined by *TD* is described by the *i*-th <column definition> in *TD*. The name and the data type or domain of the column are specified by the <column name> and <data type> or <domain name>, respectively.

   Note: The definitions of the first *n* columns of the table defined by *TD* are determined as described in the Syntax Rules of Subclause 11.3, "<table definition>".

4) Let *C* be the <column name> of the <column definition>.

5) If, in a <stored column>, neither a <data type> nor a <domain name> is specified, then there shall be a <column constraint definition> containing a <references specification> whose referenced column is of data type object identifier. A <data type> REF is implicit.

6) If <stored column> is specified and neither CONSTANT nor UPDATABLE is specified, then UPDATABLE is implicit.

7) If <virtual column> is specified, then:

   a) Let *T* be the abstract data type *TD* or the type of the table *TD* specifying the <virtual column>.

   b) If <get function> is not specified, then a <get function> is effectively specified with a <routine name> of *C*.

   c) If <set function> is not specified and *T* contains a member function named SET_*C*, then a <set function> is effectively specified with a <routine name> of SET_*C*.

   d) The <encapsulation level> contained in the <virtual column> shall be the same as that of the get function.

   e) *T* shall contain a member function whose <routine name> is the same as the <routine name> contained in the <get function>.

   f) The <get function> shall take a single argument whose type shall be *T*. Let *RT* be the result type of the <get function>. If <data type> is not specified, then the data type *RT* is implicit.

   g) Case:

      i) If a <set function> is not specified, then neither CONSTANT nor UPDATABLE shall be specified; if READ ONLY is not specified, then READ ONLY is implicit.

      ii) If a <set function> is specified, then:

         1) READ ONLY shall not be specified.

         2) If neither CONSTANT nor UPDATABLE is specified, then UPDATABLE is implicit.

         3) *T* shall contain a member function whose <routine name> is the same as the <routine name> contained in the <set function>.

         4) The <set function> shall take two arguments: the first for the target and the second for the value. The type of the target shall be *TD*.

            Note: When parameter passing semantics have been clarified, *TD* must be specified to allow update, *e.g.*, as IN OUT.

            Does this mean ''INOUT'', or ''IN OUT''?

8) If <domain name> is specified, then let *D* be the domain identified by the <domain name>.

9) The data type of the column is

   Case:

   a) If <data type> is specified, then that data type.

   b) Otherwise, the data type of *D*.

10) If the data type of the column is character string, then the collation of the column is

Case:

a) If <collate clause> is specified, then the collation specified by that <collate clause>.

b) If <domain name> is specified, then the collation of *D*, if any.

c) Otherwise, the default collation of the character set of the column.

Note: The character set of a column is determined by its data type.

11) If a <data type> is specified, then:

a) Let *DT* be the <data type>.

b) The data type of the column is *DT*. If <virtual column> is specified, then *DT* shall be the same data type as *RT*.

c) If DT is CHARACTER or CHARACTER VARYING and does not specify a <character set specification>, then the <character set specification> specified or implicit in the <schema character set specification> of the <schema definition> that created the schema identified by the <schema name> immediately contained in the <table name> of the containing <table definition> or <alter table statement> is implicit.

d) If *DT* is a <character string type> that identifies a character set that specifies a <collate clause> and the <column definition> does not contain a <collate clause>, then the <collate clause> of the <character string type> is implicit in the <column definition>.

12) If <collate clause> is specified, then data type shall be a character string type.

13) If a <column constraint definition> is specified, then let *CND* be the <constraint name definition> if one is specified and let *CND* be a zero-length string otherwise; let *CA* be the <constraint attributes> if specified and let *CA* be a zero-length string otherwise. The <column constraint definition> is equivalent to a <table constraint definition> as follows:

Case:

a) If a <column constraint definition> is specified that contains the <column constraint> NOT NULL, then it is equivalent to a <table constraint definition> that contains the following <table constraint>:

CND CHECK (*C* IS NOT NULL) *CA*

b) If a <column constraint definition> is specified that contains a <unique specification>, then it is equivalent to a <table constraint definition> that contains the following <table constraint>:

CND <unique specification> (*C*) *CA*

Note: The <unique specification> is defined in Subclause 11.7, "<unique constraint definition>".

c) If a <column constraint definition> is specified that contains a <references specification>, then it is equivalent to a <table constraint definition> that contains the following <table constraint>:

CND FOREIGN KEY (*C*) <references specification> *CA*

Note: The <references specification> is defined in Subclause 11.8, "<referential constraint definition>".

d) If a <column constraint definition> is specified that contains a <check constraint definition>, then it is equivalent to a <table constraint definition> that contains the following <table constraint>:

CND CHECK (<search condition>) *CA*

Each <column reference> directly contained in the <search condition> shall reference column *C*.

14) If <domain name> is specified or implied, then <null clause> shall not be specified.

15) If a <column definition> is contained in a <table definition> *TD* that specifies a <subtable clause>, then all descriptors of an inherited column are inherited from the descriptor of the column in the supertable from which the column was inherited.

16) If <column definition> of a stored column is contained in an <abstract data type definition> or in a <table definition> that has an abstract data type, then:

a) Let *C* be the <column name> of the column.

b) Let *DT* be the data type of the column.

c) Let *ADTN* be the <abstract data type name> of the containing <abstract data type definition> or <table definition>.

d) The containing <abstract data type definition> or <table definition> implicitly contains the <routine declaration>:

[<encapsulation level>] ACTOR FUNCTION *C* (:*ADTN ADTN*) RETURNS *DT*
RETURN :*ADTN.C*;
END FUNCTION

where the <encapsulation level> of the FUNCTION *C* is identical to the <encapsulation level> of the column *C*.

Access Rules

1) If <domain name> is specified, then the applicable privileges shall include USAGE on *D*.

2) If a <data type> is specified that is an abstract data type *A*, then the applicable privileges shall include DATA TYPE USAGE on *A*.

3) If <null class name> is specified, then the applicable <privileges> shall include USAGE.

General Rules

1) A <column definition> defines a column in a table.

2) The <collate clause> specifies the default collating sequence for the column. If <collate clause> is not specified, then the default collating sequence is that used for comparisons of *Coercible* coercibility attribute, as defined in Subclause 8.2, "<comparison predicate>".

3) If the <column definition> specifies <data type>, then a data type descriptor is created that described the data type of the column being defined.

4) The degree of the table or abstract data type *T* being defined in the containing <table definition>, <temporary table declaration> or <abstract data type definition> or altered by the containing <alter table statement> is increased by 1.

5) A column descriptor is created that describes the column being defined. The column descriptor includes the encapsulation level of the column and an indication of whether the column is READ ONLY, CONSTANT, or UPDATABLE and, if <default clause> is specified, the value of its <default option>.

   The name included in the column descriptor is <column name>. If the <column definition> specifies <data type>, then the column descriptor includes the data type descriptor of the column and the null class of the column; otherwise, the column descriptor includes the name of the domain of the column. The ordinal position included in the column descriptor is equal to the degree of *T*.

   If the <column definition> contains a <collate clause>, then the <collation name> of the <collate clause> is included in the column descriptor. The column descriptor includes the nullability characteristic of the column, determined according to the rules in Subclause 4.14, "Columns and attributes".

   If the column is a virtual column, then the descriptor will include the name of the get function and the name of the set function (if any) for that virtual column.

   The column descriptor is included in the table descriptor for *T*.

6) Case:

   a) If <domain name> is specified or implied, then the null class of this column is the null class of the domain specified by <domain name>.

   b) Otherwise, the null class of this column is <null class name>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <column definition> shall not contain an <encapsulation level>.

   b) A <column definition> shall not contain a <null clause>.

   ANSI Only–SQL3

   c) A <column definition> shall not contain a <virtual column>.

   d) A <column definition> shall not contain <constant or updatable>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <collate clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <column definition> shall not contain a <domain name>.

   b) A <column constraint> shall not contain a <referential triggered action>.

c) Conforming Entry SQL language shall not contain any <constraint name definition>.

11.5 <default clause>

Function

Specify the default for a column or domain.

Format

```
<default clause> ::=
        { DEFAULT | <assignment operator> } <default option>

<default option> ::=
        <literal>
      | <datetime value function>
      | USER
      | CURRENT_USER
      | SESSION_USER
      | SYSTEM_USER
      | NULL [ <left paren> <null state> <right paren> ]
      | <function invocation>
```

Syntax Rules

1) The subject data type of a <default clause> is the data type specified in the descriptor identified by the containing <column definition>, <domain definition>, <SQL variable declaration>, <alter column definition>, or <alter domain definition>.

ANSI Only—caused by ISO changes not yet considered by ANSI

2) If USER is specified, then CURRENT_USER is implicit.

3) If the subject data type of the <default clause> is an abstract data type, then <default option> shall not specify a <literal>, a <datetime value function>, USER, nor SYSTEM USER; furthermore, if <default option> specifies <function invocation>, then it shall identify a function that returns a value whose type is that of the subject data type.

4) Case:

   a) If a <literal> is specified, then:

      Case:

      i) If the subject data type is character string, then the <literal> shall be a <character string literal>. If the length of the subject data type is fixed, then the length in characters of the <character string literal> shall not be greater than the length of the subject data type. If the length of the subject data type is variable, then the length in characters of the <character string literal> shall not be greater than the maximum length of the subject data type. The <literal> shall have the same character repertoire as the subject data type.

ii) If the subject data type is bit string, then the <literal> shall be a <bit string literal> or a <hex string literal>. If the length of the subject data type is fixed, then the length in bits of the <bit string literal> or <hex string literal> shall not be greater than the length of the subject data type. If the length of the subject data type is variable, then the length in bits of the <bit string literal> or <hex string literal> shall not be greater than the maximum length of the subject data type.

iii) If the subject data type is exact numeric, then the <literal> shall be a <signed numeric literal> that simply contains an <exact numeric literal>. There shall be a representation of the value of the <literal> in the subject data type that does not lose any significant digits.

iv) If the subject data type is approximate numeric, then the <literal> shall be a <signed numeric literal>.

v) If the subject <data type> is an enumerated type, then the <literal> shall be an <enumeration literal> of that <enumerated type>.

vi) If the subject data type is datetime, then the <literal> shall be a <datetime literal> and shall contain the same <datetime field>s as the subject data type.

vii) If the subject data type is interval, then the <literal> shall be an <interval literal> and shall contain the same <interval qualifier> as the subject data type.

viii) If the subject <data type> is boolean, then the <literal> shall be a <boolean literal>.

ISO Only–SQL3

b) If USER is specified, then the subject data type shall be fixed-length character string with a length not less than 128 characters. The character set of the subject data type shall be the implementation-defined default character set that is the default for character string data types that do not specify a CHARACTER SET.

c) If CURRENT_USER, SESSION_USER, or SYSTEM_USER is specified, then the subject data type shall be character string with character set SQL_TEXT. If the length of the subject data type is fixed, then its length shall not be less than 128 characters. If the length of the subject data type is variable, then its maximum length shall not be less than 128 characters.

d) If <datetime value function> is specified, then the subject data type shall be datetime with the same datetime type as the datetime data type of the <datetime value function>.

e) A <null state>, if specified, shall be a valid null state of the null class of this column.

ANSI Only–SQL3

f) If <function invocation> is specified, then the subject data type shall be the same as the return data type of the function.

5) If the subject <data type> of the <default clause> is the object identifier type, then <default option> shall not specify a <literal>, a <datetime value function>, USER, nor SYSTEM_USER; furthermore, if <default option> specifies <function invocation>, then it shall identify a function that returns a value whose datatype is object identifier.

ANSI Only–SQL3

6) If the subject <data type> of the <default clause> is a collection type, then <default option> shall not specify a <datetime value function>, USER, CURRENT_USER, SESSION_USER, or SYSTEM_USER; furthermore, if <default option> specifies <literal> or <function invocation>, then that <default option> shall identify a value or function that returns a value, respectively, whose data type is that collection type.

Access Rules

None.

General Rules

1) The default value inserted in the column descriptor, if the <default clause> is to apply to a column, or in the domain descriptor, if the <default clause> is to apply to a domain, is the <default option>. If the subject data type is bit string with fixed length, the <default clause> specifies a <bit string literal>, and the length of the <bit string literal> is less than the fixed length of the column, then a completion condition is raised: *warning—implicit zero-bit padding*.

2) The default value of a stored column is

   Case:

   a) If the column descriptor of a column includes a default value derived from a <default option>, then the value in the column descriptor;

   b) If the column descriptor includes a domain name that identifies a domain descriptor that includes a default value derived from a <default option>, then the value in the domain descriptor;

   c) Otherwise, the general null value.

3) When a default value is required for a column in a row of a table (during execution of an ANSI <alter table statement>,
<insert statement>, <udpate statement: positioned>, or <update statement: searched>), the

default value for the column is derived from the default value in the column descriptor or domain descriptor for the column as follows:

a) If the column or domain descriptor contains NULL without a <null state>, then the general null value.

b) If the column or domain descriptor specifies NULL with a <null state>, then the null value corresponding to that <null state>.

c) If the column or domain descriptor contains a <literal>, then

Case:

i) If the subject data type is numeric, then the numeric value of the <literal>.

ii) If the subject data type is character string with variable length, then the value of the <literal>.

iii) If the subject data type is character string with fixed length, then the value of the <literal>, extended as necessary on the right with <space>s to the length in characters of the subject data type.

iv) If the subject data type is bit string with variable length, then the value of the <literal>.

v) If the subject data type is bit string with fixed length, then the value of the <literal> extended as necessary on the right with 0-valued bits to the length of the subject data type.

vi) If the subject data type is datetime or interval, then the value of the <literal>.

vii) If the subject data type is boolean, then the value of the <literal>.

viii) If the subject data type is an enumerated type, then the value of the <literal>.

ANSI Only–SQL3

ix) If the subject data type is a collection type, then the value of the <literal>.

ISO Only–SQL3

d) If the column or domain descriptor specifies USER, then the value obtained by an evaluation of USER at the time that the default value is required.

e) If the column or domain descriptor specifies CURRENT_USER, SESSION_USER, or SYSTEM_USER, then

Case:

i) If the subject data type is character string with variable length, then the value obtained by an evaluation of CURRENT_USER, SESSION_USER, or SYSTEM_USER at the time that the default value is required.

ii) If the subject data type is character string with fixed length, then the value obtained by an evaluation of CURRENT_USER, SESSION_USER, or SYSTEM_USER at the time that the default value is required, extended as necessary on the right with <space>s to the length in characters of the subject data type.

f) If the column or domain descriptor contains a <datetime value function>, then the value of an evaluation of the <datetime value function> at the time that the default value is required.

g) If the column or domain descriptor contains a <function invocation>, then the value of an invocation of that external function or SQL function at the time that the default value is required.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <default option> that specifies NULL shall not contain a <null state>.

   b) A <default option> shall not be a <function invocation>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <default option> shall not specify a <datetime value function>, SYSTEM_USER, SESSION_USER, or CURRENT_USER.

11.6 <table constraint definition>

Function

Specify an integrity constraint.

Format

```
<table constraint definition> ::=
       [ <constraint name definition> ]
       <table constraint> [ <constraint attributes> ]

<table constraint> ::=
         <unique constraint definition>
       | <referential constraint definition>
       | <check constraint definition>
```

Syntax Rules

1) If <constraint attributes> is not specified, then INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

2) If <constraint name definition> is not specified, then a <constraint name definition> that contains an implementation-dependent <constraint name> is implicit. The assigned <constraint name> shall obey the Syntax Rules of an explicit <constraint name>.

3) If a <table constraint definition> *TCD* is contained in a <table definition> that specifies a <subtable clause>, then every <table constraint definition> specified in the <table definition> of every supertable of a subtable is part of the description of the subtable. The constraints imposed by those <table constraint definition>s may be further constrained by *TCD*.

Access Rules

None.

General Rules

1) A <table constraint definition> defines a table constraint.

2) A table constraint descriptor is created that describes the table constraint being defined. The table constraint descriptor includes the <constraint name> contained in the explicit or implicit <constraint name definition>.

   The table constraint descriptor includes an indication of whether the constraint is deferrable or not deferrable and whether the initial constraint mode of the constraint is *deferred* or *immediate*.

   Case:

   a) If <unique constraint definition> is specified, then the table constraint descriptor is a unique constraint descriptor that includes an indication of whether it was defined with PRIMARY KEY or UNIQUE, and the names of the unique columns specified in the <unique column list>.

b) If <referential constraint definition> is specified, then the table constraint descriptor is a referential constraint descriptor that includes the names of the referencing columns specified in the <referencing columns> and the names of the referenced columns and referenced table specified in the <referenced table and columns>, the value of the <match type>, if specified, and the <referential triggered actions>, if specified.

c) If <check constraint definition> is specified, then the table constraint descriptor is a table check constraint descriptor that includes the <search condition>.

3) If the <table constraint> is a <check constraint definition>, then let *SC* be the <search condition> immediately contained in the <check constraint definition> and let *T* be the table name included in the corresponding table constraint descriptor; the table constraint is not satisfied if and only if

EXISTS (SELECT * FROM *T* WHERE NOT (*SC*))

is true.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall contain no <constraint name definition>.

11.7 <unique constraint definition>

Function

Specify a uniqueness constraint for a table.

Format

```
<unique constraint definition> ::=
       <unique specification>
         [ <left paren> <unique column list> <right paren> ]

<unique specification> ::=
       UNIQUE | PRIMARY KEY

<unique column list> ::= <column name list>
```

Syntax Rules

1) Let *T* be the table identified by the containing <table definition> or <alter table statement>. Let *TN* be the <table name> of *T*.

2) If the <unique constraint definition> contains a non-empty <unique column list>, then let *UCL* be that <unique column list>.

   Case:

   a) If the <unique specification> specifies PRIMARY KEY, then let *SC* be the <search condition>:

      UNIQUE (SELECT *UCL* FROM *TN*)
       AND
      (*UCL*) IS NOT NULL

   b) Otherwise, let *SC* be the <search condition>:

      UNIQUE (SELECT *UCL* FROM *TN*)

3) If <unique column list> is not specified, then the <unique specification> shall specify PRIMARY KEY.

4) If the <unique column list> is omitted, then an empty <unique column list> is implicit.

5) If an empty <unique column list> is specified, then the following <check constraint definition> is implicit:

   CHECK ((SELECT COUNT (*) FROM *T*) <= 1)

6) Each <column name> in the <unique column list> shall identify a stored column of *T*, and the same column shall not be identified more than once.

7) If any <column name> in the <unique column list> identifies a column whose data type is an abstract data type, then the <abstract data type definition> of that abstract data type shall contain an <equals clause> that does not specify NONE.

8) If the <unique specification> specifies PRIMARY KEY, then for each <column name> in the explicit or implicit <unique column list> for which NOT NULL is not specified, NOT NULL is implicit in the <column definition>.

9) A <table definition> shall specify at most one implicit or explicit <unique constraint definition> that specifies PRIMARY KEY.

10) If a <unique constraint definition> that specifies PRIMARY KEY is contained in an <add table constraint definition>, then the table identified by the <table name> immediately contained in the containing <alter table statement> shall not have a unique constraint that was defined by a <unique constraint definition> that specified PRIMARY KEY.

11) The set of columns in the <unique column list> shall be distinct from the unique columns of any other unique constraint descriptor that is included in the base table descriptor of *T*.

12) If a <unique constraint definition> is contained in a <table definition> *TD* that specifies a <subtable clause>, then the <unique constraint definition> shall only specify UNIQUE and shall not specify PRIMARY KEY.

Access Rules

None.

General Rules

1) A <unique constraint definition> defines a unique constraint.

   Note: Subclause 10.8, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked.

2) Let *designated columns* be the columns identified by the <column name>s of the <unique column list>.

3) *T* is constrained to contain no rows that are duplicates with respect to the designated columns. Two rows are duplicates if the value of each designated column in the first row is non-null and is equal to the value of the corresponding column in the second row.

   Note: If the <unique column list> is empty, then all rows are duplicated, so *T* is constrained to hold not more than one row.

4) The unique constraint is not satisfied if and only if

   EXISTS (SELECT * FROM *TN* WHERE NOT (*SC*))

   is true.

****Editor's Note****

The preceding Rule was inserted here for alignment with SQL-92, but the Editor is uncertain whether it is appropriate, given the earlier General Rules in this Subclause. Guidance is requested.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <unique constraint definition> shall specify a <unique column list>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If PRIMARY KEY or UNIQUE is specified, then the <column definition> for each column whose <column name> is in the <unique column list> shall specify NOT NULL.

11.8 <referential constraint definition>

Function

Specify a referential constraint.

Format

```
<referential constraint definition> ::=
       FOREIGN KEY [ <left paren> <referencing columns> <right paren> ]
         <references specification>

<references specification> ::=
       REFERENCES [ PENDANT ] <referenced table and columns>
         [ MATCH <match type> ]
         [ <referential triggered action> ]

<match type> ::=
         FULL
       | PARTIAL

<referencing columns> ::=
       <reference column list>

<referenced table and columns> ::=
       <table name> [ <left paren> [ <reference column list> ] <right paren> ]

<reference column list> ::= <column name list>

<referential triggered action> ::=
         <update rule> [ <delete rule> ]
       | <delete rule> [ <update rule> ]

<update rule> ::= ON UPDATE <referential action>

<delete rule> ::= ON DELETE <referential action>

<referential action> ::=
         CASCADE
       | SET NULL [ <left paren> <null state> <right paren> ]
       | SET DEFAULT
       | RESTRICT
       | NO ACTION
```

Syntax Rules

1) Let *referencing table* be the table identified by the containing <table definition> or <alter table statement>. Let *referenced table* be the table identified by the <table name> in the <referenced table and columns>. Let *referencing columns* be the column or columns identified by the <reference column list> in the <referencing columns> and let *referencing column* be one such column.

2) <reference column list> shall not contain the <column name> of a virtual column.

3) Case:

   a) If the <referenced table and columns> specifies a <reference column list>, then the set of column names of that <reference column list> shall be equal to the set of column names in the unique columns of a unique constraint of the referenced table. Let *referenced columns* be the column or columns identified by that <reference column list> and let *referenced column* be one such column. Each referenced column shall identify a column of the referenced table and the same column shall not be identified more than once.

   b) If the <referenced table and columns> does not specify a <reference column list>, then the table descriptor of the referenced table shall include a unique constraint that specifies PRIMARY KEY. Let *referenced columns* be the column or columns identified by the unique columns in that unique constraint and let *referenced column* be one such column. The <referenced table and columns> shall be considered to implicitly specify a <reference column list> that is identical to that <unique column list>.

4) The table constraint descriptor describing the <unique constraint definition> whose <unique column list> identifies the referenced columns shall indicate that the unique constraint is not deferrable.

5) The referenced table shall be a base table.

   Case:

   a) If the referencing table is a persistent base table, then the referenced table shall be a persistent base table.

   b) If the referencing table is a global temporary table, then the referenced table shall be a global temporary table.

   c) If the referencing table is a created local temporary table, then the referenced table shall be either a global temporary table or a created local temporary table.

   d) If the referencing table is a declared local temporary table, then the referenced table shall be either a global temporary table, a created local temporary table or a declared local temporary table.

6) If the referenced table is a temporary table with ON COMMIT DELETE ROWS specified, then the referencing table shall specify ON COMMIT DELETE ROWS.

7) For a <references specification> specifying PENDANT, the applicable <privileges> for the <table name> shall also include DELETE.

   Note: The *applicable privileges* for a <table name> are defined in Subclause 10.5, "<privileges>".

8) Each referencing column shall identify a column of the referencing table, and the same column shall not be identified more than once.

9) The <referencing columns> shall contain the same number of <column name>s as the <referenced table and columns>. The *i*-th column identified in the <referencing columns> corresponds to the *i*-th column identified in the <referenced table and columns>. The data type of each referencing column shall be comparable to the data type of the corresponding referenced column.

10) If a <referential constraint definition> does not specify any <update rule>, then an <update rule> with a <referential action> of NO ACTION is implicit.

11) If a <referential constraint definition> does not specify any <delete rule>, then a <delete rule> with a <referential action> of NO ACTION is implicit.

12) If a <referential action> specifies <null state>, then each referencing column shall have the same null class and the null state identified by <null state> shall be in that null class.

13) If the <references specification> specifies PENDANT, then no <referential action> for that <referential constraint definition> shall specify SET DEFAULT.

14) If <references specification> specifies PENDANT, then <match type> shall not specify PARTIAL.

15) If <references specification> specifies PENDANT or <delete rule> specifies ON DELETE CASCADE, then the referencing table shall not be a table identified by the <table name> of a trigger descriptor whose <trigger event> specifies DELETE.

Access Rules

1) The applicable privileges shall include REFERENCES for each referenced column. For a <references specification> specifying PENDANT, the applicable <privileges> shall also include DELETE.

General Rules

1) A <referential constraint definition> defines a referential constraint.

   Note: Subclause 10.8, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked.

2) Let R_f be the referencing columns and let R_t be the referenced columns in the referenced table *T*. The referencing table and the referenced table satisfy the referential constraint if and only if:

   Case:

   a) R_f and R_t are empty and

      0 < (SELECT COUNT (*) FROM *T*)

      is true.

   b) A <match type> is not specified and for each row of the referencing table, the <match predicate>

      R_f MATCH (SELECT R_t FROM *T*)

      is true.

   c) PARTIAL is specified and for each row of the referencing table, the <match predicate>

      R_f MATCH PARTIAL (SELECT R_t FROM *T*)

      is true.

   d) FULL is specified and for each row of the referencing table, the <match predicate>

      R_f MATCH FULL (SELECT R_t FROM *T*)

      is true.

3) If PENDANT is specified, then:

   a) For a given row in the referencing table, let *pendant reference* designate an instance in which all R_f are non-null.

   b) Let *number of pendant paths* be the number of pendant references to the same referenced row in a referenced table from all referencing rows in all base tables.

   c) For every row in *T*, the number of pendant paths is equal to or greater than 1.

4) Case:

   a) If R_f and R_t are empty, then all rows in the referencing table are *matching rows* for the only row in *T*.

   b) If <match type> is not specified or if FULL is specified, then for a given row in the referenced table, let *matching rows* be all rows in the referencing table whose referencing column values equal the corresponding referenced column values for the referential constraint.

   c) If PARTIAL is specified, then:

      i) For a given row in the referenced table, let *matching rows* be all rows in the referencing table that have at least one non-null referencing column value and whose non-null referencing column values equal the corresponding referenced column values for the referential constraint.

      ii) For a given row in the referenced table, let *unique matching rows* be all matching rows for that given row that are matching rows only to the given row in the referenced table for the referential constraint. For a given row in the referenced table, let *non-unique matching rows* be all matching rows for that given row that are not unique matching rows for that given row for the referential constraint.

5) Case:

   a) If the cursor mode of the current SQL-transaction is *cascade on*, then for each row of the referenced table, its matching rows, unique matching rows, and non-unique matching rows are determined immediately prior to the execution of any <SQL procedure statement>. No new matching rows are added during the execution of that <SQL procedure statement> unless the association is a result of the direct update of a referencing row with pendant reference.

   b) If the cursor mode of the current SQL-transaction is *cascade off*, then matching rows, unique matching rows, and non-unique matching rows are evaluated immediately prior to the first execution of a <delete statement: positioned> or <update statement: positioned> in the current set-processing mode session of the current SQL-transaction. During execution of that <SQL procedure statement>, a referenced row may not enter into an association with a matching row that was not in effect immediately prior to the first execution of a <delete statement: positioned> or <update statement: positioned> in the current set-processing mode session of the current SQL-transaction unless the association is a result of the direct update of a referencing row with pendant reference.

   The association between a referenced row and a non-unique matching row is dropped during the execution of that SQL-statement if the referenced row is either marked for deletion or updated to a distinct value on any referenced column that corresponds to a non-null referencing column. This occurs immediately after such a mark for deletion or update of the referenced row. Unique matching rows and non-unique matching rows for a referenced row are evaluated immediately

after dropping the association between that referenced row and a non-unique matching row and immediately after creating or dropping any association between a referencing row with a pendant reference and that referenced row.

6) If a <delete rule> is specified and a row of the referenced table that has not previously been marked for deletion is marked for deletion, then

Case:

a) If <match type> is not specified or if FULL is specified, then

Case:

i) If the <delete rule> specifies CASCADE, then all matching rows are marked for deletion.

ii) If the <delete rule> specifies SET NULL, then in all matching rows each referencing column is set to the general null value.

iii) If the <delete rule> specifies SET DEFAULT, then in all matching rows each referencing column is set to the default value specified in the General Rules of Subclause 11.5, "<default clause>".

iv) If the <delete rule> specifies RESTRICT and there exists some matching row, then an exception condition is raised: *restrict violation*.

b) If PARTIAL is specified, then

Case:

i) If the <delete rule> specifies CASCADE, then all unique matching rows are marked for deletion.

ii) If the <delete rule> specifies SET NULL, then in all unique matching rows each referencing column is set to the general null value.

iii) If the <delete rule> specifies SET DEFAULT, then in all unique matching rows each referencing column is set to the default value specified in the General Rules of Subclause 11.5, "<default clause>".

iv) If the <delete rule> specifies RESTRICT and there exists some unique matching row, then an exception condition is raised: *restrict violation*.

Note: Otherwise, the <referential action> is not performed.

7) If an <update rule> is specified and a non-null value of a referenced column in the referenced table is updated to a value that is distinct from the current value of that column, then

Case:

a) If <match type> is not specified or if FULL is specified, then

Case:

i) If the <update rule> specifies CASCADE, then in all matching rows the referencing column that corresponds with the referenced column is updated to the new value of the referenced column.

ii) If the <update rule> specifies SET NULL, then

Case:

1) If <match type> is not specified, then in all matching rows the referencing column that corresponds with the referenced column is set to the general null value.

2) If <match type> specifies FULL, then in all matching rows each referencing column is set to the general null value.

iii) If the <update rule> specifies SET DEFAULT, then in all matching rows the referencing column that corresponds with the referenced column is set to the default value specified in the General Rules of Subclause 11.5, "<default clause>".

iv) If the <update rule> specifies RESTRICT and there exists some matching row, then an exception condition is raised: *restrict violation*.

b) If PARTIAL is specified, then

Case:

i) If the <update rule> specifies CASCADE, then for each unique matching row that contains a non-null value in the referencing column *C1* that corresponds with the updated referenced column *C2*, *C1* is updated to the new value *V* of *C2*, provided that, in all updated rows in the referenced table that formerly had, in the same SQL-statement, that unique matching row as a matching row, the values in *C2* have all been updated to a value that is not distinct from *V*. Otherwise, an exception condition is raised: *triggered data change violation*.

Note: Because of the Rules of Subclause 8.2, "<comparison predicate>", on which the definition of ''distinct'' relies, the values in *C2* may have been updated to values that are not distinct, yet are not identical. Which of these non-distinct values is used for the cascade operation is implementation-dependent.

ii) If the <update rule> specifies SET NULL, then in all unique matching rows that contain a non-null value in the referencing column that corresponds with the updated column, that referencing column is set to the general null value.

iii) If the <update rule> specifies SET DEFAULT, then in all unique matching rows that contain a non-null value in the referencing column that corresponds with the updated column, that referencing column is set to the default value specified in the General Rules of Subclause 11.5, "<default clause>".

iv) If the <update rule> specifies RESTRICT and there exists some unique matching row, then an exception condition is raised: *restrict violation*.

Note: Otherwise, the <referential action> is not performed.

8) If a referencing row with a pendant reference is inserted into a referencing table:

Case:

a) If no prior pendant reference to the referenced row exists, then the number of pendant paths for that referenced row is 1 and an association between the referencing row and that referenced row is created.

b) If the number of pendant paths for the referenced row is greater than or equal to 1, then the number of pendant paths for that referenced row is increased by 1 and an association between the referencing row and that referenced row is created.

9) If a referencing row with a pendant reference is marked for deletion from a referencing table, then the number of pendant paths for the referenced row is decreased by 1.

10) If a referencing column of a referencing table contained in a <references specification> specifying PENDANT is updated:

Case:

a) If the value of the referencing column is changed from a null value to a pendant reference, then cases under the General Rule beginning ''If a referencing row with a pendant reference is inserted into a referencing table:'' prevail.

b) If the value of the referencing column changes from a pendant reference to a null value, then the number of pendant paths for the referenced row is decreased by 1 and the association between the referencing row and that referenced row is dropped.

c) If the value of the referencing column changes from one pendant reference to another pendant reference, then the number of pendant paths for the referenced row of the former pendant reference is decreased by one, the association between the referencing row and that referenced row is dropped, and cases under the General Rule beginning ''If a referencing row with a pendant reference is inserted into a referencing table:'' prevail for the latter pendant reference.

11) Let *T* be the table identified by the <table name> of a <delete statement: searched> or an <update statement: searched>. If a mark for deletion of or update to *T* causes the number of pendant paths to some referenced row to decrease to 0, then a recursive search for each referenced row whose number of pendant paths is equal to 0 is effectively performed after all qualifying rows of *T* are marked for deletion or updated, and after all <referential action>s triggered by the marks for deletion or updates to *T* are performed. A recursive search comprises the following actions: If a referenced row with 0 pendant paths is found, it is marked for deletion; each subsequent search for another referenced row with 0 pendant paths is effectively performed after all <referential action>s triggered by the deletion of the previous referenced row are performed.

12) Case:

a) If the cursor mode of the current SQL-transaction is *cascade on*, then:

i) If any attempt is made within an SQL-statement to update some data item to a value that is distinct from the value to which that data item was previously updated within the same SQL-statement, then an exception condition is raised: *triggered data change violation*.

ii) If a data item in an object row is an <object column> of an <update statement: positioned> or <update statement: searched>, and there is any attempt within the same SQL-statement to delete the row containing that data item, then an exception condition is raised: *triggered data change violation*.

Note: Rows that are explicitly updated by an <update statement: positioned> or an <update statement: searched> are prohibited from later being (implicitly) deleted by a triggered PENDANT action.

b) If the cursor mode of the current SQL-transaction is *cascade off*, then:

i) If any attempt is made during the current set-processing mode session of the current SQL-transaction to update some data item to a value that is distinct from the value to which that data item was previously updated within the same set-processing mode

session of the same SQL-transaction, then an exception condition is raised: *triggered data change violation*.

ii) If a data item in an object row is an <object column> of an <update statement: positioned> or <update statement: searched>, and there is any attempt within the same set-processing mode session of the same SQL-transaction to delete the row containing that data item, then an exception condition is raised: *triggered data change violation*.

Note: Rows that are explicitly updated by an <update statement: positioned> or an <update statement: searched> are prohibited from later being (implicitly) deleted by a triggered PENDANT action.

13) If an <update rule> attempts to update a row that has been deleted by any <delete statement: positioned> that identifies some cursor *CR* that is still open or updated by any <update statement: positioned> that identifies some cursor *CR* that is still open or if a <delete rule> attempts to mark for deletion such a row, then a completion condition is raised: *warning—cursor operation conflict*.

14) Case:

a) If the cursor mode of the current SQL-transaction is *cascade on*, then all rows that are marked for deletion are effectively deleeted at the end of the SQL-statement, prior to the checking of any integrity constraints.

b) If the cursor mode of the current SQL-transaction is *cascade off*, then:

i) The deletion of all rows that are marked for deletion is effectively deferred until the execution of a <commit statement> or the execution of a <close statement>.

ii) All <referential action>s and pendant actions are effectively deferred until the execution of a <commit statement> or the execution of a <close statement>.

15) When a row is effectively deleted, if the row is in a table that contains a default destructor function, then that function is invoked with its <argument> being the
ANSI abstract data type instance representing that
ISO object identifier of the
row.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <references specification> shall not contain PENDANT.

b) If a <referenced table and columns> specifies a <left paren>, then it shall also specify a <reference column list>.

c) A <referential action> shall not be RESTRICT.

d) A <referential action> shall not be NO ACTION.

e) The data type of each referencing column shall be the same as the data type of the corresponding referenced column.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) A <references specification> shall not specify MATCH.

b) A <referential triggered action> shall not contain an <update rule>.

c) The order of the column names in a <reference column list> shall be the same as the order of column names of the corresponding unique constraint of the referenced table.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <referential constraint definition> shall not contain a <referential triggered action>.

11.9 <check constraint definition>

Function

Specify a condition for the SQL-data.

Format

```
<check constraint definition> ::=
       CHECK <left paren> <search condition> <right paren>
```

Syntax Rules

1) The <search condition> shall not contain a <target specification>.

2) The <search condition> shall not contain a <set function specification> that is not contained in a <subquery>.

3) If <check constraint definition> is contained in a <table definition> or <alter table statement>, then let *T* be the table identified by the containing <table definition> or <alter table statement>.

   Case:

   a) If *T* is a persistent base table, or if the <check constraint definition> is contained in a <domain definition> or <alter domain statement>, then no <table reference> generally contained in the <search condition> shall reference a temporary table.

   b) If *T* is a global temporary table, then no <table reference> generally contained in the <search condition> shall reference a table other than a global temporary table.

   c) If *T* is a created local temporary table, then no <table reference> generally contained in the <search condition> shall reference a table other than either a global temporary table or a created local temporary table.

   d) If *T* is a declared local temporary table, then no <table reference> generally contained in the <search condition> shall reference a persistent base table.

4) If the <check constraint definition> is contained in a <table definition> that defines a temporary table and specifies ON COMMIT PRESERVE ROWS or a <temporary table declaration> that specifies ON COMMIT PRESERVE ROWS, then no <subquery> in the <search condition> shall reference a temporary table defined by a <table definition> or a <temporary table declaration> that specifies ON COMMIT DELETE ROWS.

5) The <search condition> shall not generally contain a <datetime value function> or a <value specification> that is CURRENT_USER, SESSION_USER, or SYSTEM_USER.

6) The <search condition> shall not generally contain a <query specification> or a <query expression> that is possibly non-deterministic.

Access Rules

1) Let *TN* be any <table name> referenced in the <search condition>.

   Case:

   a) If a <column name> is contained in the <search condition>, then the applicable privileges shall include REFERENCES for each <column name> of the table identified by *TN* contained in the <search condition>.

   b) Otherwise, the applicable privileges shall include REFERENCES for at least one column of the table identified by *TN*.

General Rules

1) A <check constraint definition> defines a check constraint.

   Note: Subclause 10.8, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked. The General Rules that control the evaluation of a check constraint can be found in either Subclause 11.6, "<table constraint definition>", or Subclause 11.24, "<domain definition>", depending on whether it forms part of a table constraint or a domain constraint.

ANSI Only–SQL3

2) Case:

   a) If the <check constraint definition> is contained in a <table definition>, then it is applicable to that table.

   b) If the <check constraint definition> is contained in a <domain definition>, then it is applicable to every value required to be in that domain.

   c) If the <check constraint definition> is contained in an <abstract data type body>, then it is applicable to every instance of an abstract data type defined by or generated from that <abstract data type body>.

3) If the character representation of the <search condition> cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning—search condition too long for information schema*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) The <search condition> contained in a <check constraint definition> shall not contain a <subquery>.

   b) The REFERENCES privilege is not required for <check constraint definition> access.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.10 <alter table statement>

Function

Change the definition of a table.

Format

```
<alter table statement> ::=
```

ANSI Only--SQL3

```
      ALTER <table type> <table name> <alter table action>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
      ALTER TABLE <table name> <alter table action>
```

```
<alter table action> ::=
        <add column definition>
      | <alter column definition>
      | <drop column definition>
      | <add supertable clause>
      | <drop supertable clause>
      | <add table constraint definition>
      | <drop table constraint definition>
```

Syntax Rules

1) Let *T* be the table identified by the <table name>.

2) The schema identified by the explicit or implicit schema name of the <table name> shall include the descriptor of *T*.

ANSI Only–SQL3

3) If the specified <table type> is not TABLE, then it shall be the same as that in the descriptor for *T*.

4) The scope of the <table name> is the entire <alter table statement>.

5) *T* shall be a base table.

6) *T* shall not be a declared local temporary table.

7) If *T* has a descriptor containing the name of an abstract data type *ADT*, then the <alter table action> shall be either <add table constraint definition>, or <drop table constraint definition> that shall not contain the <constraint name> of any constraint in the descriptor of *ADT*.

****Editor's Note****

The preceding Rule prevents *any* table from having a column added, dropped, or altered or a supertable added or dropped, since *all* tables are currently defined to have an abstract data type associated with them. See Possible Problem **274** in the Editor's Notes.

Access Rules

1) The current <authorization identifier> shall be equal to the the <authorization identifier> that owns the schema identified by the <schema name> of the table identified by <table name>.

General Rules

1) The base table descriptor of *T* is modified as specified by <alter table action>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <alter table action> shall not be a <add supertable clause>.

   b) An <alter table action> shall not be a <drop supertable clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain an <alter table statement>.

11.11 <add column definition>

Function

Add a column to a table.

Format

```
<add column definition> ::=
       ADD [ COLUMN ] <column definition>
```

Syntax Rules

1) If the <column definition> does not contain an <encapsulation level>, then PUBLIC is implicit.

Access Rules

None.

General Rules

1) The column defined by the <column definition> is added to *T*.

2) If the <column definition> is for a stored column, then let *C* be the column added to *T*. Every value in *C* is the default value for *C*.

   Note: The default value of a column is defined in Subclause 11.5, "<default clause>".

   Note: The addition of a column to a table has no effect on any existing <query expression> included in a view descriptor or <search condition> included in constraint descriptor because any implicit <column reference>s in these clauses are replaced by explicit <column reference>s when the clause is originally evaluated. See the Syntax Rules of Subclause 7.13, "<query expression>".

ANSI Only–SQL3

3) If *T* is the subject table in a trigger descriptor that contains an UPDATE trigger event with an implicit trigger column list, then *C* is included in the trigger column list of this trigger descriptor. The effect of adding *C* to the implicit trigger column list will be such that an update of *C* in *T* will cause the triggered actions to be executed.

ISO Only—caused by ANSI changes not yet considered by ISO

4) If *T* is the <table name> of a trigger descriptor *TD* that contains an UPDATE event without an explicitly-specified <trigger column list>, then *C* is included in the implied <trigger column list> of *TD*. The effect of adding *C* to the implied <trigger column list> is such that an update of *C* in *T* will cause the <triggered action> to be executed.

5) For every table privilege descriptor that specifies *T* and a privilege of SELECT, UPDATE, INSERT or REFERENCES, a new column privilege descriptor is created that specifies *T*, the same action, grantor, and grantee, and the same grantability, and specifies the <column name> of the <column definition>.

6) In all other respects, the specification of a <column definition> in an <alter table statement> has the same effect as specification of the <column definition> in the <table definition> for *T* would have had. In particular, the degree of *T* is increased by 1 and the ordinal position of that column is equal to the new degree of *T* as specified in the General Rules of Subclause 11.4, "<column definition>".

7) If *T* is a subtable, then the descriptor of the column specified in the <column definition> defines an originally defined column of *T*.

8) If *T* is a supertable, then an <add column definition>, without further Access Rule checking, is effectively performed for each of its subtables, thereby adding the column as an inherited column in these subtables.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain an <add column definition>.

11.12 <alter column definition>

Function

Change a column and its definition.

Format

```
<alter column definition> ::=
       ALTER [ COLUMN ] <column name> <alter column action>

<alter column action> ::=
         <set column default clause>
       | <drop column default clause>
       | <drop column domain clause>
```

Syntax Rules

1) Let *T* be the table identified in the containing <alter table statement>.

2) Let *C* be the column identified by the <column name>.

3) *C* shall be a stored column of *T*.

4) *C* shall not have <column name> OID.

5) If <drop domain domain clause> is specified, then *C* shall be based on a domain.

Access Rules

None.

General Rules

1) The column descriptor of *C* is modified as specified by <alter column action>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <drop column domain clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain an <alter column definition>.

11.13 <set column default clause>

Function

Set the default clause for a column.

Format

```
<set column default clause> ::=
       SET <default clause>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *C* be the column identified by the <column name> in the containing <alter column definition>.

2) The default value specified by the <default clause> is placed in the column descriptor of *C*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <set column default clause>.

11.14 <drop column default clause>

Function

Drop the default clause from a column.

Format

```
<drop column default clause> ::=
       DROP DEFAULT
```

Syntax Rules

1) Let *C* be the column identified by the <column name> in the containing <alter column definition>.

2) The descriptor of *C* shall include a default value.

Access Rules

None.

General Rules

1) The default value is removed from the column descriptor of *C*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop column default clause>.

11.15 <drop column domain clause>

Function

Drop the domain from a column.

Format

```
<drop column domain clause> ::=
       DROP DOMAIN [ <constraint disposition> [ <constraint name list> ] ]
```

Syntax Rules

1) Let *D* be the domain of the column identified by the <column name> immdeiately contained in the containing <alter column definition> and let *DN* be the <domain name> of *D*.

2) If <constraint disposition> is not specified, then KEEP COLUMN CONSTRAINT is implicit.

3) If <constraint disposition> is specified and <constraint name list> is not specified, then ALL is implicit.

4) If a <constraint name list> is specified, then every <constraint name> in the <constraint name list> shall be the <constraint name> of a domain constraint descriptor included in the domain descriptor of *D*.

5) If ALL is specified or implicit, then it is equivalent to a <constraint name list> that contains each <constraint name> of every domain constraint descriptor included in the domain descriptor of *D*.

Access Rules

1) Let *UA* be the <authorization identifier> that owns the schema identified by the <schema name> of the table identified by <table name>.

General Rules

1) Let *C* be any column descriptor that includes *DN*, let *T* be the table described by the table descriptor that includes *C*, and let *TN* be the table name of *T*. *C* is modified as follows:

   a) *DN* is removed from *C*. A copy of the data type descriptor of *D* is included in *C*.

   b) If *C* does not include a <default clause> and the domain descriptor of *D* includes a <default clause>, then a copy of the <default clause> of *D* is included in *C*.

   c) If DROP COLUMN CONSTRAINT is specified, then let the *excluded constraint list* be the <constraint name>s in the implicit or explicit <constraint name list>. If KEEP COLUMN CONSTRAINT is specified or implied, then let the *excluded constraint list* be the <constraint name> of each domain constraint descriptor included in the domain descriptor of *D* that does not occur in the implicit or explicit <constraint name list>.

For every domain constraint descriptor included in the domain descriptor of *D* whose <constraint name> is not contained in the excluded constraint list:

i) Let *TCD* be a <table constraint definition> consisting of a <constraint name definition> whose <constraint name> is implementation-dependent, whose <table constraint> is derived from the <check constraint definition> of the domain constraint descriptor by replacing every instance of VALUE by the <column name> of *C*, and whose <constraint attributes> are the <constraint attributes> of the domain constraint descriptor.

ii) If the applicable privileges of *UA* include all of the privileges necessary for *UA* to successfully execute the <add table constraint definition>

ALTER TABLE *TN* ADD *TCD*

then the following <table constraint definition> is effectively executed with a current <authorization identifier> of *UA*:

ALTER TABLE *TN* ADD *TCD*

d) If *C* does not include a collation and the <domain definition> of *D* includes a collation, then

i) Let *CCN* be the <collation name> of the collation.

ii) If the applicable privileges for *UA* contain USAGE on *CCN*, then *CCN* is added to *C* as the <collation name>.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) Conforming Full SQL language shall not contain any <drop column domain clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.16 <drop column definition>

Function

Destroy a column.

Format

```
<drop column definition> ::=
       DROP [ COLUMN ] <column name> <drop behavior>
```

Syntax Rules

1) Let *T* be the table identified by the <table name> in the containing <alter table statement> and let *TN* be the name of *T*.

2) Let *C* be the column identified by the <column name> *CN*.

3) *C* shall be a column of *T* and *C* shall not be the only column of *T*.

4) If RESTRICT is specified, then *C* shall not be referenced in the <query expression> of any view descriptor or in the <search condition> of any constraint descriptor other than a table constraint descriptor that contains references to no other column and that is included in the table descriptor of *T*.

   Note: A <drop column definition> that does not specify CASCADE will fail if there are any references to that column resulting from the use of CORRESPONDING, NATURAL, SELECT * (except where contained in an exists predicate>), or REFERENCES without a <reference column list> in its <referenced table and columns>.

5) *C* shall not have <column name> OID.

6) If RESTRICT is specified, then *CN* shall not be explicitly referenced in a <trigger definition>.

   Note: If CASCADE is specified, then any such dependent object will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

7) If RESTRICT is specified, then *T* shall not be a supertable.

8) If *T* is a subtable, then the <column name> shall not specify a <column name> of an inherited column.

Access Rules

None.

General Rules

1) Let *TR* be the <trigger name> of any <trigger definition> that contains *CN*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

   DROP TRIGGER *TR*

ANSI Only–SQL3

2) If *T* is the subject table of a trigger descriptor *TD* that contains an UPDATE trigger event with an implicit trigger column list, then *C* is removed from the trigger column list of *TD*.

ISO Only—caused by ANSI changes not yet considered by ISO

3) If *T* is the <table name> of a trigger descriptor *TD* that contains an UPDATE event without an explictly-specified <trigger column list>, then *C* is removed from the implied <trigger column list> of *TD*.

4) Let *VN* be the name of any view that contains a reference to column *C* of table *T*. The following <drop view statement> is effectively executred with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

   DROP VIEW *VN* CASCADE

5) If *T* is a supertable, then all columns inherited from *CN* in any subtable of *T* are also destroyed.

6) Let *ST* be the <table name> of any subtable of *T*. The following <alter table statement> is effectively executed without further Access Rule checking:

   ALTER TABLE *ST* DROP *CN* CASCADE

7) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

   REVOKE INSERT(*CN*), UPDATE(*CN*), REFERENCES(*CN*) ON TABLE *TN* FROM *A* CASCADE

8) If the column is not based on a domain, then its data type descriptor is destroyed.

9) If *C* is a stored column, then the data associated with *C* is destroyed.

10) The descriptor of *C* is removed from the descriptor of *T*.

11) The identified column and its descriptor are destroyed.

12) The degree of *T* is reduced by 1. The ordinal position of all columns having an ordinal position greater than the ordinal position of *C* is reduced by 1.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop column definition>.

11.17 <add supertable clause>

Function

Add a supertable clause to a table.

Format

```
<add supertable clause> ::=
       ADD <supertable clause>
```

Syntax Rules

1) Let *T* be the table identified in the containing <alter table statement>.

2) Let *TG* be the supertable identified by the <supertable name> in the <supertable clause>.

3) *T* shall not have *TG* as a subtable.

4) *TG* shall not have *T* as a direct subtable.

5) There shall be exactly one maximal supertable in the subtable family of *T* with the addition of *TG* to that family.

Access Rules

None.

General Rules

1) The name of *TG* is included as a direct supertable in the table descriptor of *T*.

2) The column descriptors of *TG* that are not already inherited by *T* are appended to the descriptor of *T* as inherited columns. For every <member renaming element> in the <supertable clause>, the <column name> of the inherited column identified by <supertable member name> is replaced by the <subtable member name>.

3) For every newly inherited column in *T*, an <add column definition> is effectively performed for that column without further Access Rule checking.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <add supertable clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain an <add table constraint definition>.

11.18 <drop supertable clause>

Function

Drop a supertable clause from a table.

Format

```
<drop supertable clause> ::=
       DROP <supertable clause> <drop behavior>
```

Syntax Rules

1) Let *T* denote the table identified in the containing <alter table statement>.

2) Let *TG* denote the supertable identified by the <supertable name> in the <supertable clause>.

3) *T* shall be a direct subtable of *TG*.

Access Rules

1) For every column of *TG* that is not also inherited by *T* from some other supertable, a <drop column definition> is effectively performed for that column without further Access Rule checking. If <drop behavior> is specified, then the same <drop behavior>is implicitly specified in each of the <drop column definition>s.

General Rules

1) The name of the supertable is effectively dropped from the table descriptor of *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop supertable clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop table constraint definition>.

11.19 <add table constraint definition>

Function

Add a constraint to a table.

Format

```
<add table constraint definition> ::=
       ADD <table constraint definition>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *T* be the table identified by the <table name> in the containing <alter table statement>.

2) The table constraint descriptor for the <table constraint definition> is included in the table descriptor for *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain an <add table constraint definition>.

11.20 <drop table constraint definition>

Function

Destroy a constraint on a table.

Format

```
<drop table constraint definition> ::=
       DROP CONSTRAINT <constraint name> <drop behavior>
```

Syntax Rules

1) Let *T* be the table identified by the <table name> in the containing <alter table statement>.

2) The <constraint name> shall identify a table constraint *TC* of *T*.

3) If *TC* is a unique constraint and there exists a referential constraint *RC* whose referenced table is *T* and whose referenced columns are the unique columns of *TC*, then *RC* is said to be *dependent on TC*.

4) The constraint shall not be PRIMARY KEY on any column with <column name> OID.

5) If RESTRICT is specified, then no table constraint shall be dependent on *TC*.

   Note: If CASCADE is specified, then any such dependent object will be dropped by the effective execution of the <alter table statement> specified in the General Rules of this Subclause.

Access Rules

None.

General Rules

1) Let *TCN2* be the <constraint name> of any table constraint that is dependent on *TC* and let *T2* be the <table name> of the table descriptor that includes *TCN2*. The following <alter table statement> is effectively executed without further Access Rule checking:

   ALTER TABLE *T2* DROP CONSTRAINT *TCN2* CASCADE

2) The descriptor of *TC* is removed from the descriptor of *T*.

3) The identified table constraint and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop table constraint definition>.

11.21 <drop table statement>

Function

Destroy a table.

Format

```
<drop table statement> ::=
```

ANSI Only--SQL3

```
DROP <table type> <table name> <drop behavior>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
DROP TABLE <table name> <drop behavior>
```

Syntax Rules

1) Let *T* be the table identified by the <table name> and let *TN* be that <table name>.

2) The schema identified by the explicit or implicit schema name of the <table name> shall include the descriptor of *T*.

3) *T* shall be a base table.

ANSI Only–SQL3

4) If the specified <table type> is not TABLE, then it shall be the same as that in the descriptor for *T*.

5) *T* shall not be a declared local temporary table.

6) If RESTRICT is specified, then *T* shall not be referenced in the <query expression> of any view descriptor or the <search condition> of any constraint descriptor.

Note: If CASCADE is specified, then such referenced objects will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the table identified by *TN*.

General Rules

1) Let *ST* be the <table name> of any subtable of *T*. The following <drop table statement> is effectively executed without further Access Rule checking:

   DROP TABLE *ST* CASCADE

2) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

   REVOKE ALL PRIVILEGES ON *TN* FROM *A* CASCADE

3) The identified base table and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <drop table statement>.

11.22 <view definition>

Function

Define a viewed table.

Format

```
<view definition> ::=
       CREATE VIEW <table name> [ <left paren> <view column list> <right paren> ]
         AS <query expression>
         [ WITH [ <levels clause> ] CHECK OPTION ]

<levels clause> ::=
       CASCADED | LOCAL

<view column list> ::= <column name list>
```

Syntax Rules

1) The <query expression> shall not contain a <target specification>.

2) If a <view definition> is contained in a <schema definition> and the <table name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

3) The schema identified by the explicit or implicit schema name of the <table name> shall not include a table descriptor whose table name is <table name>.

4) No <table reference> generally contained in the <query expression> shall identify the viewed table defined by <view definition> nor any temporary view nor any declared local temporary table.

5) If the <query expression> is updatable, then the viewed table is an updatable table. Otherwise, it is a read-only table.

6) If the <query expression> is a <query specification> that contains a <group by clause> or a <having clause> that is not contained in a <subquery>, then the viewed table defined by the <view definition> is a *grouped view*.

7) If any two columns in the table specified by the <query expression> have the same <column name>, or if any column of that table has an implementation-dependent name, then a <view column list> shall be specified.

8) The same <column name> shall not be specified more than once in the <view column list>.

9) The number of <column name>s in the <view column list> shall be the same as the degree of the table specified by the <query expression>.

10) No column in the table specified by <query expression> shall have a coercibility attribute of *No collating sequence*.

Note: The coercibility attribute is described in Subclause 4.2.3, "Rules determining collating sequence usage".

Note: The coercibility attribute for references to the column is defined in Subclause 6.6, "<column reference> and <object reference>".

11) If WITH CHECK OPTION is specified with no <levels clause>, then a <levels clause> of CASCADED is implicit.

12) If WITH*nul1* LOCAL CHECK OPTION is specified, then the <query expression> shall not generally contain a <query expression> *QE* or a <query specification> *QS* that is *possibly non-deterministic* unless *QE* or *QS* is generally contained in a viewed table that is a root underlying table of the <query expression>.

If WITH CASCADED CHECK OPTION is specified, then the <query expression> shall not generally contain a <query expression> or <query specification> that is *possibly non-deterministic*.

13) Let *V* be the view defined by the <view definition>. The underlying columns of every *i*-th column of *V* are the underlying columns of the *i*-th column of the <query expression> and the underlying columns of *V* are the underlying columns of the <query expression>.

Access Rules

1) If a <view definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <table name>.

General Rules

1) A view descriptor *VD* is created that describes *V*. The view descriptor includes the <table name>, the <query expression>, column descriptors taken from the table specified by the <query expression>, and an indication of whether WITH CHECK OPTION was specified. If a <view column list> is specified, then the <column name> of the *i*-th column of the view is the *i*-th <column name> in that <view column list>. Otherwise, the <column name>s of the view are the <column name>s of the table specified by the <query expression>.

ANSI Only–SQL3

The data type of the objects in the viewed table and the indication of whether the viewed table is a multiset table, a set table, or a list table are taken from the table specified by the <query expression>.

2) Let *VN* be the <table name>. Let *QE* be the <query expression>. If a <view column list> is specified, then let *VCL* be the <view column list> preceded by a <left paren> and followed by a <right paren>; otherwise, let *VCL* be the empty string.

Case:

a) When *VN* is immediately contained in some SQL-schema statement, it identifies the view descriptor *VD*.

b) Otherwise, *VN* references the same table as the <table reference>:

(*QE*) AS *VN VCL*

3) Let *A* be the <authorization identifier> that owns *V*.

4) A set of privilege descriptors is created that defines the privilege SELECT on this table to *A* and SELECT for each column of *V* to *A*. This privilege is grantable if and only if the applicable SELECT <privileges> on all <table name>s contained in the <query expression> are grantable. The grantor of this privilege descriptor is set to the special grantor value ''_SYSTEM''.

ANSI Only—caused by ISO changes not yet considered by ANSI

5) If *V* is updatable, then let *T* be the leaf underlying table of the <query expression>.

6) For *i* ranging from 1 to the number of distinct leaf underlying tables of the <query expression> of *V*, let RT_i be the <table name>s of those tables. For every column *CV* of *V*:

   a) Let CRT_{ij}, for *j* ranging from 1 to the number of columns of RT_i that are underlying columns of *CV*, be the <column name>s of those columns.

   b) If *A* has REFERENCES(CRT_{ij}) for all *i* and for all *j*, and *A* has REFERENCES on some column of RT_i for all *i*, then a privilege descriptor is created that defines the privilege REFERENCES (*CV*) on *V* to *A*. That privilege is grantable if and only if the REFERENCES privileges on all of the columns CRT_{ij} are grantable. The grantor of that privilege descriptor is set to the special grantor value ''_SYSTEM''.

ISO Only–SQL3

7) Let *DT* be derived table that is the result of the <query expression>. If *V* is updatable, then for each leaf underlying table *LUT* of *DT*, if *DT* is fully updatable with respect to *LUT*, then:

   a) A set of privilege descriptors is created that defines the privileges INSERT, UPDATE and DELETE on *V* that are applicable privileges on *LUT* to *A*.

   b) For each column *CV* of *V* that has a counterpart *CLUT* in *LUT*, a set of privilege descriptors is created that defines the privileges INSERT(*CV*) and UPDATE(*CV*) on *V*, where the privileges INSERT(*CLUT*) and UPDATE(*CLUT*) on *LUT* are applicable privileges to *A*, respectively.

   c) A privilege on *V* is grantable if and only if the corresponding privilege on *LUT* is grantable.

ANSI Only—caused by ISO changes not yet considered by ANSI

8) If *V* is updatable, then:

   a) A set of privilege descriptors is created that defines the privileges INSERT, UPDATE, and DELETE on *V* that are applicable privileges on *T* to *A*. A privilege on *V* is grantable if and only if the corresponding privilege on *T* is grantable.

b) For every updatable column in *V*:

i) There is a corresponding column in *T* from which the column of *V* is derived. Let *CV* and *CT* be the <column name>s of the corresponding columns of *V* and *T* respectively.

ii) A set of privilege descriptors is created that defines the privileges INSERT(*CV*) and UPDATE(*CV*) on *V*, where the privileges INSERT(*CT*) and UPDATE(*CT*) on *T* are the applicable privileges to *A*, respectively. A privilege on *V* is grantable if and only if the corresponding privilege on *T* is grantable.

The grantor of these privilege descriptors is set to the special grantor value ''_SYSTEM''.

****Editor's Note****

The fact that REFERENCES privileges are not propogated to views has been identified as a Possible Problem. See Possible Problem **159** in the Editor's Notes.

ANSI Only—caused by ISO changes not yet considered by ANSI

9) If *V* is updatable, then let *TLEAF* be the leaf generally underlying table of *V*. For every row in *V* there is a corresponding row in *TLEAF* from which the row of *V* is derived and for each column in *V* there is a corresponding column in *TLEAF* from which the column of *V* is derived. The insertion of a row into *V* is an insertion of a corresponding row into *TLEAF*. The deletion of a row from *V* is a deletion of the corresponding row in *TLEAF*. The updating of a column of a row in *V* is an updating of the corresponding column of the corresponding row in *TLEAF*.

10) Let *V1* be a view. *V1 spans V1*. *V1 spans* a view *V2* if *V2* is a generally underlying table of *V1*.

11) An *update operation* is an <insert statement>, <update statement: positioned>, <update statement: searched>, <dynamic update statement: positioned>, or <preparable dynamic update statement: positioned>. An update operation on a view *V* is an update operation whose <table name> identifies *V*.

12) If a view *V1* spans a view *VA* described by a view descriptor that includes WITH CHECK OPTION and an update operation on *V1* would result in a row that would not appear in the result of *VA*, then

a) If the view descriptor of *VA* includes CASCADED, then an exception condition is raised: *with check option violation*.

b) If the view descriptor of *VA* includes LOCAL and the update operation would result in a row that would appear in the simply underlying table of the simply underlying table of the <query expression> contained in *VA*, then an exception condition is raised: *with check option violation*.

****Editor's Note****

This will have to be worded a little differently for SQL3 because views with more than one ''S.U.T.'' are updatable there. Please supply the appropriate wording.

13) Validation of a WITH CHECK OPTION constraint is effectively performed at the end of the each operation.

14) If the character representation of the <query expression> cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning—query expression too long for information schema*.

15) Case:

a) If the cursor mode of the current SQL-transaction is *cascade off*, then the row validation for the WITH CHECK OPTION is effectively performed at the end of the <insert statement>, <update statement: positioned>, or <update statement: searched>.

b) If the cursor mode of the current SQL-transaction is *cascade on*, then the row validation for the WITH CHECK OPTION is effectively deferred until implicitly enacted by execution of a <commit statement> or implicitly enacted by execution of a <close statement>.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <levels clause>, but the effect shall be that defined for a <levels clause> of CASCADED.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) The <query expression> in a <view definition> shall be a <query specification>.

11.23 <drop view statement>

Function

Destroy a view.

Format

```
<drop view statement> ::=
       DROP VIEW <table name> <drop behavior>
```

Syntax Rules

1) Let *V* be the table identified by the <table name> and let *VN* be that <table name>. The schema identified by the explicit or implicit schema name of *VN* shall include the descriptor of *V*.

2) *V* shall be a viewed table.

3) If RESTRICT is specified, then *V* shall not be referenced in the <query expression> of any view descriptor, or the <search condition> of any assertion descriptor or constraint descriptor, or in any trigger descriptor.

   Note: If CASCADE is specified, then any such dependent object will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the table identified by *VN*.

General Rules

1) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

   REVOKE ALL PRIVILEGES ON *VN* FROM *A* CASCADE

2) The identified view and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop view statement>.

11.24 <domain definition>

Function

Define a domain.

Format

```
<domain definition> ::=
       CREATE DOMAIN <domain name> [ AS ] <data type>
         [ <default clause> ]
         [ <domain constraint>... ]
         [ <collate clause> ]
         [ <null clause> ]

<domain constraint> ::=
       [ <constraint name definition> ]
       <check constraint definition> [ <constraint attributes> ]
```

Syntax Rules

1) If a <domain definition> is contained in a <schema definition>, and if the <domain name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>. The schema identified by the explicit or implicit schema name of the <domain name> shall not include a domain descriptor whose domain name is <domain name> nor an abstract data type descriptor whose abstract data type name is <domain name>.

2) If <data type> specifies CHARACTER or CHARACTER VARYING and does not specify <character set specification>, then the character set name of the default character set of the schema identified by the implicit or explicit <schema name> of <domain name> is implicit.

3) If <data type> specifies a <character string type> that identifies a character set that has a default collation and the <domain definition> does not directly contain a <collate clause>, then the collation of the <character string type> is the implicit collation of the domain.

4) Let *D1* be some domain. *D1* is *in usage by* a domain constraint *DC* if and only if the <search condition> of *DC* generally contains the <domain name> either of *D1* or of some domain *D2* such that *D1* is in usage by some domain constraint of *D2*. No domain shall be in usage by any of its own constraints.

5) The <search condition> of a <domain constraint> shall not contain a <domain name>.

6) If <collate clause> is specified, then <data type> shall be a character string type.

7) For every <domain constraint> specified:

   a) If <constraint attributes> is not specified, then INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

   b) If <constraint name definition> is not specified, then a <constraint name definition> that contains an implementation-dependent <constraint name> is implicit. The assigned <constraint name> shall obey the Syntax Rules of an explicit <constraint name>.

• 1 Rule deleted.

Access Rules

1) If a <domain definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <domain name>.

2) If <null class name> is specified, then the applicable <privileges> shall include USAGE.

General Rules

1) A <domain definition> defines a domain.

   Note: Subclause 10.8, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked.

2) A data type descriptor is created that describes the data type of the domain being created.

3) A domain descriptor is created that describes the domain being created. The domain descriptor contains the name of the domain, the data type descriptor of the data type, the <collation name> of the <collate clause> if the <domain definition> contains a <collate clause>, the value of the <default clause> if the <domain definition> immediately contains <default clause>, and a domain constraint descriptor for every immediately contained <domain constraint>.

4) A privilege descriptor is created that defines the USAGE privilege on this domain to the <authorization identifier> of the <schema> or <module> in which the <domain definition> appears. This privilege is grantable if and only if the applicable privileges include a grantable REFERENCES privilege for each <column reference> included in the domain descriptor and a grantable USAGE privilege for each <collation name>, <character set name>, and <translation name> contained in the contained in the <search condition> of any domain constraint descriptor included in the domain descriptor. The grantor of the privilege descriptor is set to the special grantor value ''_SYSTEM''.

5) Let *DSC* be the <search condition> included in some domain constraint descriptor *DCD*. Let *D* be the name of the domain whose descriptor includes *DCD*. Let *T* be the name of some table whose descriptor includes some column descriptor with column name *C* whose domain name is *D*. Let *CSC* be a copy of *DSC* in which every instance of the <general value specification> VALUE is replaced by *C*.

6) The domain constraint specified by *DCD* for *C* is not satisfied if and only if

   EXISTS (SELECT * FROM *T* WHERE NOT (*CSC*))

   is true.

7) The null class of this domain is <null class name>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <collate clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <domain definition>.

11.25 <alter domain statement>

Function

Change a domain and its definition.

Format

```
<alter domain statement> ::=
       ALTER DOMAIN <domain name> <alter domain action>

<alter domain action> ::=
         <set domain default clause>
       | <drop domain default clause>
       | <add domain constraint definition>
       | <drop domain constraint definition>
```

Syntax Rules

1) Let *D* be the domain identified by <domain name>. The schema identified by the explicit or implicit schema name of the <domain name> shall include the descriptor of *D*.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of <domain name>.

General Rules

1) The domain descriptor of *D* is modified as specified by <alter domain action>.

   Note: The changed domain descriptor of *D* is applicable to every column that is dependent on *D*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <alter domain statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.26 <set domain default clause>

Function

Set the default value in a domain.

Format

```
<set domain default clause> ::= SET <default clause>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *D* be the domain identified by the <domain name> in the containing <alter domain statement>.

2) The default value specified by the <default clause> is placed in the domain descriptor of *D*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <set domain default clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.27 <drop domain default clause>

Function

Remove the default clause of a domain.

Format

```
<drop domain default clause> ::= DROP DEFAULT
```

Syntax Rules

1) Let *D* be the domain identified by the <domain name> in the containing <alter domain statement>.

2) The descriptor of *D* shall contain a default value.

Access Rules

None.

General Rules

1) Let *C* be the set of columns whose column descriptors contain the domain descriptor of *D*.

2) For every column belonging to *C*, if the column descriptor does not already contain a default value, then the default value from the domain descriptor of *D* is placed in that column descriptor.

3) The default value is removed from the domain descriptor of *D*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <drop domain default clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.28 <add domain constraint definition>

Function

Add a constraint to a domain.

Format

```
<add domain constraint definition> ::=
       ADD <domain constraint>
```

Syntax Rules

1) Let *D* be the domain identified by the <domain name> in the <alter domain statement>.

2) Let *D1* be some domain. *D1* is *in usage by* a domain constraint *DC* if and only if the <search condition> of *DC* generally contains the <domain name> either of *D1* or of some domain *D2* such that *D1* is in usage by some domain constraint of *D2*. No domain shall be in usage by any of its own constraints.

Access Rules

None.

General Rules

1) The constraint descriptor of the <domain constraint> is added to the domain descriptor of *D*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <add domain constraint definition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.29 <drop domain constraint definition>

Function

Destroy a constraint on a domain.

Format

```
<drop domain constraint definition> ::=
       DROP CONSTRAINT <constraint name>
         [ <constraint disposition> ]
```

Syntax Rules

1) Let *D* be the domain identified by the <domain name> *DN* in the containing <alter domain statement>.

2) Let *CD* be any column descriptor that includes *DN*, let *T* be the table desribed by the table desriptor that includes *CD*, and let *TN* be the <table name> of *T*.

3) If <constraint disposition> is not specified, then DROP COLUMN CONSTRAINT is implicit.

4) Let *DC* be the descriptor of the constraint identified by <constraint name>.

5) *DC* shall be included in the domain descriptor of *D*.

Access Rules

1) Let *UA* be the <authorization identifier> that owns the schema identified by the <schema name> of a table identified by *TN*.

General Rules

1) If KEEP COLUMN CONSTRAINT is specified, then for every domain constraint descriptor included in the domain descriptor of *D* and for each column *C* that is based on domain *D*:

   a) Let *TCD* be a <table constraint definition> comprising a <constraint name definition> whose <constraint name> is implementation-dependent, whose <table constraint> is derived from the <check constraint definition> of the domain constraint descriptor by replacing every instance of VALUE by the <column name> of *C*, and whose <constraint attributes> are the <constraint attributes> of the domain constraint descriptor.

   b) If the applicable privileges of *UA* include all of the privileges necessary for *UA* to successfully execute the <add table constraint definition>

      ALTER TABLE *TN* ADD *TCD*

      then the following <table constraint definition> is effectively executed with a current <authorization identifier> of *UA*:

      ALTER TABLE *TN* ADD *TCD*

2) The constraint descriptor of *DC* is removed from the domain descriptor of *D*.

3) The constraint *DC* and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <constraint disposition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <drop domain constraint definition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.30 <drop domain statement>

Function

Destroy a domain.

Format

```
<drop domain statement> ::=
       DROP DOMAIN <domain name> <drop behavior>
         [ <constraint disposition> [ <constraint name list> ] ]

<constraint disposition> ::=
         KEEP COLUMN CONSTRAINT
       | DROP COLUMN CONSTRAINT
```

- 1 production deleted.

Syntax Rules

1) Let *D* be the domain identified by <domain name> and let *DN* be that <domain name>. The schema identified by the explicit or implicit schema name of *DN* shall include the descriptor of *D*.

2) If RESTRICT is specified, then *D* shall not be referenced by any column descriptor, in the <query expression> of any view descriptor, or in the <search condition> of any constraint descriptor.

3) If RESTRICT is specified, then <constraint disposition> shall not be specified.

4) If CASCADE is specified and <constraint disposition> is not specified, then KEEP COLUMN CONSTRAINT is implicit.

5) If <constraint disposition> is specified and <constraint name list> is not specified, then ALL is implicit.

6) If a <constraint name list> is specified, then every <constraint name> in the <constraint name list> shall be the <constraint name> of a domain constraint descriptor included in the domain descriptor of *D*.

7) If ALL is specified or implicit, then it is equivalent to a <constraint name list> that contains the <constraint name> of every domain constraint descriptor included in the domain descriptor of *D*.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the domain identified by *DN*. Let *UA* be the <authorization identifier> of the current SQL-session.

General Rules

1) Let *C* be any column descriptor that includes *DN*, let *T* be the table described by the table descriptor that includes *C*, and let *TN* be the table name of *T*. *C* is modified as follows:

   a) *DN* is removed from *C*. A copy of the data type descriptor of *D* is included in *C*.

   b) If *C* does not include a <default clause> and the domain descriptor of *D* includes a <default clause>, then a copy of the <default clause> of *D* is included in *C*.

   c) If DROP COLUMN CONSTRAINT is specified, then let the *excluded constraint list* be the <constraint name>s in the implicit or explicit <constraint name list>. If KEEP COLUMN CONSTRAINT is specified or implied, then let the *excluded constraint list* be the <constraint name> of each domain constraint descriptor included in the domain descriptor of *D* that does not occur in the implicit or explicit <constraint name list>.

      For every domain constraint descriptor included in the domain descriptor of *D* whose <constraint name> is not contained in the excluded constraint list:

      i) Let *TCD* be a <table constraint definition> consisting of a <constraint name definition> whose <constraint name> is implementation-dependent, whose <table constraint> is derived from the <check constraint definition> of the domain constraint descriptor by replacing every instance of VALUE by the <column name> of *C*, and whose <constraint attributes> are the <constraint attributes> of the domain constraint descriptor.

      ii) If the applicable privileges of *UA* include all of the privileges necessary for *UA* to successfully execute the <add table constraint definition>

         ALTER TABLE *TN* ADD *TCD*

      then the following <table constraint definition> is effectively executed with a current <authorization identifier> of *UA*:

         ALTER TABLE *TN* ADD *TCD*

   d) If *C* does not include a collation and the <domain definition> of *D* includes a collation, then

      i) Let *CCN* be the <collation name> of the collation.

      ii) If the applicable privileges for *UA* contain USAGE on *CCN*, then *CCN* is added to *C* as the <collation name>.

2) If *T* is a supertable, then all subtables of *T* are destroyed.

****Editor's Note****
The preceding Rule causes all subtables of a supertable to be destroyed if a domain used by any column in the supertable is dropped. That seems an unusually harsh penalty when the data type and other attributes of the domain are ''adopted'' by the column in the supertable—couldn't the same ''adoption'' be made by the corresponding columns in the subtables? Guidance is requested.

3) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

   REVOKE USAGE ON DOMAIN *DN* FROM *A* CASCADE

****Editor's Note****

The preceding Rule and the following one seem to differ in one important respect. The following Rule (currently ANSI-only) revokes every privilege on the object granted by ''_SYSTEM'' to the owner/creator of the object, whereas the preceding Rule (currently ISO-only) revokes every privilege on the object granted by ''_SYSTEM'' to the destroyer of the object. In SQL2, the destroyer of an object must, of necessity, be the creator of the object (that is because only the owner of a schema can create and destroy objects in that schema). However, a possible SQL3 enhancement might be the definition of ''schema privileges'' that permit actors other than the owner of a schema to create and destroy objects in that schema—even to the point that an actor other than the creator of an object might be permitted to destroy the object.

The author of LON-070, point H, correctly notes that Subclause 11.36, "<drop collation statement>", already has a Rule of this (preceding) type. Putting such Rules in place everywhere proposed by LON-070, point H, is fine, but we must remember to revise the Rules if any SQL3 ''schema privilegs'' proposal is ever accepted.

This comment applies to this Subclause as well as to Subclause 11.21, "<drop table statement>", Subclause 11.23, "<drop view statement>", Subclause 11.16, "<drop column definition>", Subclause 11.36, "<drop collation statement>", Subclause 11.34, "<drop character set statement>", and Subclause 11.38, "<drop translation statement>".

The Rule (which ANSI deleted in favor of the ISO rule above):

For every privilege descriptor with a grantor of ''_SYSTEM'' that specifies USAGE on *D*, the following <revoke statement> is effectively executed as if the <authorization identifier> of the current SQL-session were ''_SYSTEM'' and without Access Rule checking:

REVOKE USAGE ON DOMAIN *DN* FROM *GRANTEE* CASCADE

where *GRANTEE* is the grantee in the privilege descriptor.

4) The identified domain is destroyed by destroying its descriptor and its data type descriptor.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <constraint disposition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop domain statement>.

11.31 <null class definition>

Function

Define a null class.

Format

```
<null class definition> ::=
       CREATE NULL CLASS <null class name> [ AS ]
           ( <null state list> )

<null state list> ::=
       <null state> [ { <comma> <null state> }... ]

<null state> ::= <identifier>
```

Syntax Rules

1) Case:

   a) If a <null class definition> is contained in a <schema definition>, and if the <null class name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

   b) If a <null class definition> is contained in a <module>, and if the <null class name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the <module>.

2) The <null class name> shall be different from the <null class name> of any other null class in the containing schema.

3) In a <null state list>, each <null state> shall be different from any other <null state> of the <null state list>.

4) A <null state list> defines an ordering of null states. Each <null state> yields a different null state value. The predefined order relation between null state values follows the order of corresponding position numbers. The position number of the value of the first listed <null state> is 2. The position number of each other null state value is one more than the position number of its predecessor in the list.

Access Rules

1) If a <null class definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <null class name>.

General Rules

1) A privilege descriptor is created that defines the USAGE privilege on this null class to the <authorization identifier> of the <schema> or <module> in which the <null class definition> appears. This privilege is grantable. The grantor of the privilege descriptor is set to the special grantor value ''_SYSTEM''.

2) The possible null states of the null class are the general null value (effectively having position number one) together with null states identified in the <null state list>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <null class definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.32 <drop null class statement>

Function

Destroy a null class.

Format

```
<drop null class statement> ::=
       DROP NULL CLASS <null class name>
```

Syntax Rules

1) Let *N* denote the null class identified by <null class name> and let *NN* be the <null class name>.

2) *N* shall not be the null class of any domain or column.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the null class identified by *NN*.

General Rules

1) For every privilege descriptor with a grantor of ''_SYSTEM'' that specifies USAGE on *N*, the following <revoke statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without Access Rule checking:

   REVOKE USAGE ON *NN* FROM *GRANTEE* CASCADE

   where *GRANTEE* is the grantee in the privilege descriptor.

2) All privilege descriptors referencing *N* are destroyed.

3) The descriptor of *N* is destroyed and *N* is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop null class statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <drop null class statement>.

11.33 <character set definition>

Function

Define a character set.

Format

```
<character set definition> ::=
       CREATE CHARACTER SET <character set name> [ AS ]
         <character set source>
         <form-of-use specification>
         [ <collate clause> | <limited collation definition> ]

<character set source> ::=
         GET <existing character set name>
             [ <plus sign> <character set source> ]
       | <left paren> <character list> <right paren>

<existing character set name> ::=
         <standard character repertoire name>
       | <implementation-defined character repertoire name>
       | <schema character set name>

<form-of-use specification> ::= <identifier>

<schema character set name> ::= <character set name>

<limited collation definition> ::=
       COLLATION FROM <collation source>

<character list> ::=
       <character specification> [ { <comma> <character specification> }... ]

<character specification> ::=
         <character string literal>
       | <ISO 10646 position>
       | <ISO 10646 character name>

<ISO 10646 position> ::=
       <val> [ <sep> <val> [ <sep> <val> [ <sep> <val> ] ] ]

<val> ::= <unsigned numeric literal>

<sep> ::= <ampersand>

<ISO 10646 character name> ::= !! See the Syntax Rules
```

Syntax Rules

1) If a <character set definition> is contained in a <schema definition> and if the <character set name> immediately contained in the <character set definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the <schema definition>.

2) The schema identified by the explicit or implicit schema name of the <character set name> shall not include a character set descriptor whose character set name is <character set name>.

3) The <standard character repertoire name>s and <implementation-defined character repertoire name>s that are supported are implementation-defined. Each character set identified by a <standard character repertoire name> or by a <implementation-defined character repertoire name> shall have associated with it a privilege descriptor that was effectively defined by the <grant statement>

   GRANT USAGE ON CHARACTER SET *CS* TO PUBLIC

   where *CS* is the <standard character repertoire name> or <implementation-defined character repertoire name>.

4) A <character string literal> directly contained in a <character specification> shall have length 1.

5) A <schema character set name> shall identify some character set descriptor.

   Note: If the <character set definition> is not specified in a <schema definition>, then <collate clause> cannot be specified, because this ''circular reference'' is permitted only when the <character set definition> and the <collation definition> are defined ''simultaneously'' in a single <schema definition>.

6) An <ISO 10646 position> shall be a valid code that identifies a character position in ISO/IEC DIS 10646.

7) An <ISO 10646 character name> shall be the name of a character as given in ISO/IEC DIS 10646 except that <space> characters are replaced by <underscore>s in the name.

8) If the <character set definition> is not specified in a <schema definition>, then <collate clause> shall not be specified.

9) If neither <collate clause> nor <limited collation definition> is specified, then the following <limited collation definition> is implicit:

   COLLATION FROM DEFAULT

10) If <limited collation definition> is specified, then <collation source> shall not be an <internal collation source>.

Access Rules

1) If a <character set definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <character set name>.

2) The applicable privileges for the <existing character set name> shall include USAGE.

General Rules

1) A <character set definition> defines a character set.

2) A character set descriptor is created for the defined character set.

3) The character set that is created contains every character in each of the character sets identified by <existing character set name>s, if specified, and in the <character set list>, if specified. Any redundant duplicate characters are deleted from the created character set.

4) If ONE_OCTET is specified, then the resulting character set is represented with 1 octet per character. If TWO_OCTET is specified, then the resulting character set is represented with 2 octets per character. If FOUR_OCTET is specified, then the resulting character set is represented with 4 octets per character.

5) A privilege descriptor is created that defines the USAGE privilege on this character set to the <authorization identifier> of the schema or <module> in which the <character set definition> appears. The grantor of the privilege descriptor is set to the special grantor value ''_SYSTEM''. This privilege is grantable.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <character set source> shall not contain a <plus sign>.

   b) A <character set source> shall not contain a <character list>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) In conforming Intermediate SQL language, <collation source> shall specify DEFAULT.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not specify any <character set definition>.

11.34 <drop character set statement>

Function

Destroy a character set.

Format

```
<drop character set statement> ::=
       DROP CHARACTER SET <character set name>
```

Syntax Rules

1) Let *C* be the character set identified by the <character set name> and let *CN* be the name of *C*.

2) The schema identified by the explicit or implicit schema name of *CN* shall include the descriptor of *C*.

3) *C* shall not be referenced in the <query expression> of any view descriptor or in the <search condition> of any constraint descriptor, nor be included in any collation descriptor or translation descriptor or schema descriptor, nor be referenced in the <module character set specification> of any module, nor be included in any column's or any domain's data type descriptor.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the character set identified by *C*.

General Rules

1) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

   REVOKE USAGE ON CHARACTER SET *CN* FROM *A* CASCADE

2) The descriptor of *C* is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall contain no <drop character set statement>.

11.35 <collation definition>

Function

Define a collating sequence.

Format

```
<collation definition> ::=
       CREATE COLLATION <collation name> FOR <character set specification>
         FROM <collation source>
           [ <pad attribute> ]

<pad attribute> ::=
         NO PAD
       | PAD SPACE

<collation source> ::=
         <collating sequence definition>
       | <translation collation>
       | <collation dictionary specification>
       | <collation routine specification>

<collation routine specification> ::=
         ROUTINE <left paren> <implementation-defined routine name>
             <left paren> <params> <right paren> <right paren>

<params> ::= !! Not yet defined

<implementation-defined routine name> ::= !! See the Syntax Rules



<collating sequence definition> ::=
         <external collation>
       | <schema collation name>
       | <internal collation source>
       | DESC <left paren> <collation name> <right paren>
       | DEFAULT

<translation collation> ::=
       TRANSLATION <translation name>
           [ THEN COLLATION <collation name> ]

<collation dictionary specification> ::=
       DICTIONARY <dictionary name> [ <plus sign> <dictionary name> ]

<dictionary name> ::=
       <quote> <implementation-defined dictionary name> <quote>

<implementation-defined dictionary name> ::= !! See the Syntax Rules

<external collation> ::=
       EXTERNAL <left paren> <quote> <external collation name> <quote> <right paren>

<schema collation name> ::= <collation name>

<external collation name> ::=
         <standard collation name>
```

```
       | <implementation-defined collation name>

<standard collation name> ::= <collation name>

<implementation-defined collation name> ::= <collation name>

<internal collation source> ::= <left paren> <collation options> <right paren>

<collation options> ::=
       <collation option> [ { <comma> <collation option> }... ]

<collation option> ::=
         USING <left paren> <collating basis> <right paren>
       | SEQUENCE <left paren> <enumerated collating sequence> <right paren>
       | MODIFY <left paren> <collating modifiers> <right paren>
       | WHEN NOT FOUND
           { IGNORE | MAX | MIN }

<enumerated collating sequence> ::=
       <collating chars> [ { <comma> <collating chars> }... ]

<collating chars> ::=
         <character specification>
       | <character range>

<character range> ::=
       <character specification> <minus sign> <character specification>

<collating modifiers> ::=
         <collating modifier> [ { <comma> <collating modifier> }... ]

<collating modifier> ::=
       <collating chars>
         { <less than operator> | <greater than operator> | <equals operator> }
       <collating chars>

<collating basis> ::=
       <collating foundation> [ { <plus sign> <collating foundation> }... ]

<collating foundation> ::=
       <collating sequence definition>
```

Note that the BNF nonterminal "<params>" is not defined.

Syntax Rules

1) If a <collation definition> is contained in a <schema definition> and if the <collation name> immediately contained in the <collation definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the <schema definition>.

2) The schema identified by the explicit or implicit schema name of the <collation name> shall not include a collation descriptor whose collation name is <collation name>.

3) A <standard collation name> shall be the name of a collation defined by a national or international standard. An <implementation-defined collation name> shall be the name of a collation that is implementation-defined.

4) The <standard collation name>s and <implementation-defined collation name>s that are supported are implementation-defined. Each collation identified by a <standard collation name> or by a <implementation-defined collation name> shall have associated with it a privilege descriptor that was effectively defined by the <grant statement>

   GRANT USAGE ON COLLATION *COLL* TO PUBLIC

   where *COLL* is the <standard collation name> or <implementation-defined collation name>.

5) A collating sequence specified by <external collation name> or <schema collation name> shall be a collating sequence that is defined for the character repertoire of the character set with which the <collation source> is associated.

6) A <schema collation name> shall be the name of a collating sequence that is defined in the schema identified by the explicit or implicit <schema name>.

7) An <implementation-defined routine name> shall be the name of an implementation-defined collation routine.

8) An <implementation-defined dictionary name> shall be the name of an implementation-defined dictionary of collation information. The structure of the dictionary is implementation-defined.

9) Each <character representation> that is contained in <collating chars> shall be a character in the character repertoire identified by <character set specification>.

10) Each collating sequence specified by <external collation> or <schema collation name> shall be a collating sequence that is defined for the character repertoire identified by <character set specification>.

11) No <collation options> shall directly contain USING, SEQUENCE, MODIFY, or WHEN NOT FOUND more than once.

12) In <collation options>, MODIFY shall not be specified first.

13) In <collation options>, if USING is specified, then SEQUENCE shall not be specified.

14) A <character range> *i* –*j* is equivalent to the <collating chars> *i,...,j*, where every character in the repertoire that appears between *i* and *j* is included in the ellipsis, in the order that the characters appear in the character repertoire.

15) If a <collation definition> does not specify <pad attribute>, then

   Case:

   a) If a <collating sequence definition> is specified that contains a <collation name> that identifies a collation for which the <collation definition> specifies NO PAD, then NO PAD is implicit.

   b) Otherwise, PAD SPACE is implicit.

16) If NO PAD is specified, then the collation is said to have the NO PAD attribute. If PAD SPACE is specified, then the collation is said to have the PAD SPACE attribute.

17) If <translation collation> is specified, then let *T* be the translation named by <translation name>. Let *C1* be the collation being defined by the <collation definition>. The source character set of *T* shall be the same as the character set of *C1*.

18) If THEN COLLATION <collation name> is specified, then let *C2* be the collation named by <collation name> in THEN COLLATION <collation name>. The target character set of *T* shall be identical to the character set of *C2*.

Access Rules

1) If a <collation definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <collation name>.

2) Let *C* be a collation identified by any <collation name> contained in <collation source>. The applicable privileges shall include USAGE on *C*.

3) If <translation name> is specified, then the applicable <privileges> shall include USAGE.

General Rules

1) A <collation definition> defines a collating sequence.

2) DEFAULT specifies that the collation is to be performed using the order of characters as they appear in the character repertoire.

3) If DESC is specified, then the collation is the reverse of that specified by <collation name>.

4) If a <collation dictionary specification> specifies more than one dictionary, then the dictionaries are effectively concatenated.

5) In a <collating modifier>, <less than operator> means that the <collating chars> on the left of the <less than operator> collate immediately preceding the <collating chars> on the right of the <less than operator>.

6) In a <collating modifier>, <greater than operator> means that the <collating chars> on the left of the <greater than operator> collate immediately following the <collating chars> on the right of the <greater than operator>.

7) In a <collating modifier>, <equals operator> means that the <collating chars> on the left of the <equals operator> collate the same as the <collating chars> on the right of the <equals operator>.

8) If <internal collation source> is specified, then:

   a) If USING is specified, then the collation is the <collating basis>.

   b) If SEQUENCE is specified, then the collation is the <enumerated collating sequence>.

   c) If MODIFY is specified, then the collation is modified by applying <collating modifiers> to the collation.

   d) If WHEN NOT FOUND is specified, then

      Case:

      i) If IGNORE is specified, then characters for which a collation is not otherwise given are ignored when using this collation.

ii) If MAX is specified, then characters for which a collation is not otherwise given are collated greater than any character for which a collation is given, but their collation is otherwise implementation-dependent.

iii) If MIN is specified, then characters for which a collation is not otherwise given are collated less than any character for which a collation is given, but their collation is otherwise implementation-dependent.

e) In a <collating basis>, multiple <collating foundation>s identify collations that are to be applied sequentially in the order that they are specified.

9) A privilege descriptor is created that defines the USAGE privilege on this collation to the current <authorization identifier>. The grantor of the privilege descriptor is set to the special grantor value ''_SYSTEM''.

10) This privilege descriptor is grantable if and only if the USAGE privilege for the current <authorization identifier> on the <character set name> contained in the <collation definition> is also grantable and if the USAGE privilege for the current <authorization identifier> on the <translation name> contained in the <translation collation>, if present, is also grantable.

11) If <translation collation> is specified, then

Case:

a) If THEN COLLATION <collation name> is specified, then let *C2* be the collating sequence named by the <collation name> in THEN COLLATION <collation name>. The collating sequence defined is obtained by effectively translating a character string using *T*, then applying the collating sequence of *C2* to the result.

b) Otherwise, the collating sequence defined is obtained by effectively translating a character string using *T*, then applying the default collating sequence for the target character set of *T*.

12) If <external collation> is specified, then the collating sequence defined is that given by:

a) If <standard collation name> is specified, then the national or international standard collation.

b) Otherwise, the implementation-defined collation.

13) A collation descriptor is created for the defined collation.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <collation source> shall not be a <collation dictionary specification>.

b) A <collation source> shall not be a <collation routine specification>.

c) A <collation sequence definition> shall not be a <internal collation source>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <collation definition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.36 <drop collation statement>

Function

Destroy a collating sequence.

Format

```
<drop collation statement> ::=
       DROP COLLATION <collation name>
         <drop behavior>
```

****Editor's Note****

The CASCADE capability for <drop collation statement> has been deferred 'til SQL3 because of certain implications: dropping a collation CASCADE will cause columns using that collation to be dropped, including the data in the column. This problem requires further thought.

Syntax Rules

1) Let *C* be the collating sequence identified by the <collation name> and let *CN* be the name of *C*.

2) The schema identified by the explicit or implicit schema name of *CN* shall include the descriptor of *C*.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the collating sequence identified by *C*.

General Rules

1) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

   For every collation descriptor *CD* that includes *CN*, *CD* is modified such that it does not include *CN*. If *CD* does not include any translation name, then *CD* is modified to indicate that it utilizes the DEFAULT collation for its character repertoire.
   For every character set descriptor *CSD* that includes *CN*, *CSD* is modified such that it does not include *CN*. If *CSD* does not include any translation name, then *CSD* is modified to indicate that it utilizes the DEFAULT collation for its character repertoire.
   For every column descriptor or domain descriptor *DD* that includes *CN*, *DD* is modified such that it does not include *CN*.

   Note: This causes the column or domain described by *DD* to revert to the default collation for its character set.

   REVOKE USAGE ON COLLATION *CN* FROM *A* CASCADE

2) The descriptor of *C* is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <drop behavior> shall not specify CASCADE.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <drop collation statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.37 <translation definition>

Function

Define a character translation.

Format

```
<translation definition> ::=
       CREATE TRANSLATION <translation name>
         FOR <source character set specification>
           TO <target character set specification>
         FROM <translation source>

<source character set specification> ::= <character set specification>

<target character set specification> ::= <character set specification>

<translation source> ::=
         <translation specification>
       | <translation routine>

<translation specification> ::=
         <external translation>
       | IDENTITY
       | <schema translation name>
       | <internal translation source>

<external translation> ::=
       EXTERNAL <left paren> <quote> <external translation name> <quote> <right paren>

<external translation name> ::=
         <standard translation name>
       | <implementation-defined translation name>

<standard translation name> ::= <translation name>

<implementation-defined translation name> ::= <translation name>

<schema translation name> ::= <translation name>

<translation routine> ::=
         ROUTINE <left paren> <implementation-defined routine name>
             <left paren> <params> <right paren> <right paren>

<params> ::= !! Not yet defined

<internal translation source> ::=
       <left paren> <translation options> <right paren>

<translation options> ::=
       <translation option> [ { <comma> <translation option> }... ]

<translation option> ::=
         USING <left paren> <translation basis> <right paren>
       | MODIFY <left paren> <translation modifiers> <right paren>

<translation modifiers> ::=
```

```
        <translation modifier> [ { <comma> <translation modifier> }... ]

<translation modifier> ::=
       <collating chars> <equals operator> <collating chars>

<translation basis> ::=
       <translation definition> [ { <asterisk> <translation definition> }... ]
```

Note that the BNF nonterminal "<params>" is not defined.

Syntax Rules

1) If a <translation definition> is contained in a <schema definition> and if the <translation name> immediately contained in the <translation definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the <schema definition>.

2) The schema identified by the explicit or implicit schema name of the <translation name> shall not include a translation descriptor whose translation name is <translation name>.

3) A <standard translation name> shall be the name of a translation defined by a national or international standard. An <implementation-defined translation name> shall be the name of a translation that is implementation-defined.

4) The <standard translation name>s and <implementation-defined translation name>s that are supported are implementation-defined. Each translation identified by a <standard translation name> or by a <implementation-defined translation name> shall have associated with it a privilege descriptor that was effectively defined by the <grant statement>

   GRANT USAGE ON TRANSLATION *TRANS* TO PUBLIC

   where *TRANS* is the <standard translation name> or <implementation-defined translation name>.

5) A <schema translation name> shall identify a translation descriptor.

6) An <implementation-defined routine name> shall be the name of an implementation-defined translation routine.

7) USING shall not be simply contained in <translation options> more than once.

8) In <translation options>, USING shall be specified before MODIFY.

9) In a <translation modifier>, the characters in the first <collating chars> shall be contained in the <target character set specification> and the characters in the second <collating chars> shall be contained in the <source character set specification>.

Access Rules

1) If a <translation definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <translation name>.

2) If <external translation name> is specified, then the applicable <privileges> shall include USAGE.

General Rules

1) A <translation definition> defines a translation.

2) IDENTITY specifies a translation that makes no changes to the characters.

3) For every character in the second <collating chars>,

   Case:

   a) If the *i*-th character in the first <collating chars> exists, then the *i*-th character in the second <collating chars> is translated into the *i*-th character in the first <collating chars>.

   b) Otherwise, the *i*-th character in the second <collating chars> is not translated.

4) If a <translation basis> specifies more than one <translation definition>, the character string is converted once for each <translation definition>, and the <translation definition>s are applied in the reverse order of their specification.

5) A translation descriptor is created for the defined translation.

6) A privilege descriptor *PD* is created that defines the USAGE privilege on this translation to the <authorization identifier> of the schema or <module> in which the <translation definition> appears. The grantor of the privilege descriptor is set to the special grantor value ''_SYSTEM''.

7) *PD* is grantable if and only if the USAGE privilege for the <authorization identifier> of the schema or <module> in which the <translation definition> appears is also grantable on every <character set name> contained in the <translation definition>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <translation source> shall not be a <translation routine>.

   b) A <translation specification> shall not be a <internal translation source>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <translation definition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.38 <drop translation statement>

Function

Destroy a character translation.

Format

```
<drop translation statement> ::=
       DROP TRANSLATION <translation name>
```

Syntax Rules

1) Let *T* be the translation identified by the <translation name> and let *TN* be the name of *T*.

2) The schema identified by the explicit or implicit schema name of *TN* shall include the descriptor of *T*.

3) *T* shall not be referenced in any <view definition>, the <query expression> included in any view descriptor or in the <search condition> included in any constraint descriptor or be included in any collation descriptor.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the translation identified by *T*.

General Rules

1) Let *CD* be any collation descriptor that includes a <translation collation> TRANSLATION *TN*. *CD* is modified by deleting that <translation collation>.

2) Let *CSD* be any <character set definition> that references *T*. *CSD* is modified by deleting any occurrences of a <translation collation> that contains *TN*.

3) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

   REVOKE USAGE ON TRANSLATION *TN* FROM *A* CASCADE

4) The descriptor of *T* is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <drop translation statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.39 <assertion definition>

Function

Specify an integrity constraint by means of an assertion and specify when the assertion is to be checked.

Format

```
<assertion definition> ::=
       CREATE ASSERTION <constraint name>
         <assertion trigger>...
         <triggered assertion> [ <constraint attributes> ]

<triggered assertion> ::=
       CHECK <left paren> <search condition> <right paren>
         [ FOR [ EACH [ ROW OF ] ] <table name> ]

<assertion trigger> ::=
         <immediate assertion trigger>
       | <deferred assertion trigger>

<deferred assertion trigger> ::= BEFORE COMMIT

<immediate assertion trigger> ::=
       AFTER { <assertion trigger statement>
         [ { <comma> <assertion trigger statement> }... ] ON <table name> }...

<assertion trigger statement> ::=
         INSERT
       | DELETE
       | UPDATE [ OF <left paren> <assertion column list> <right paren> ]

<assertion column list> ::= <column name list>
```

Syntax Rules

1) If an <assertion definition> is contained in a <schema definition> and if the <constraint name> contains a <schema name>, then that <schema name> shall be the same as the explicit or implicit <schema name> of the containing <schema definition>.

2) The schema identified by the explicit or implicit schema name of the <constraint name> shall not include a constraint descriptor whose constraint name is <constraint name>.

3) If <constraint attributes> is not specified, then INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

4) The <search condition> shall not contain a <target specification>.

5) No <query expression> in the <search condition> shall reference a temporary view or a temporary table.

6) The <search condition> shall not generally contain a <datetime value function> or a <value specification> that is
ISO USER,
CURRENT_USER, SESSION_USER, or SYSTEM_USER.

7) The <qualified identifier> of <constraint name> shall be different from the <qualified identifier> of the <constraint name> of any other constraint defined in the same schema.

8) The <search condition> shall not generally contain a <query specification> or a <query expression> that is possibly non-deterministic.

9) Let *T1* be the <table name> referenced in the <immediate assertion trigger>. *T1* shall be defined in the containing schema.

10) If <assertion column list> is specified, each <column name> in the <assertion column list> shall identify a column of *T1*.

11) If FOR is specified, then:

    a) Let *T2* be the <table name> identified in the <triggered assertion>.

    b) *T2* shall be defined in the containing schema.

    c) The <search condition> shall not contain a <set function specification> that is not contained in a <subquery>.

    d) The scope of *T2* is the entire <triggered assertion>.

Access Rules

1) If an <assertion definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <constraint name> of the <assertion definition>.

2) Let *T* be any table referenced in the <search condition> of the <assertion definition>. If *T* is a base table or a view, then

   Case:

    a) For each column *C* of *T* whose <column name> is contained in the <search condition>, the applicable privileges shall include REFERENCES for *C*.

    b) Otherwise, the applicable privileges shall include REFERENCES for at least one column of *T*.

General Rules

1) An <assertion definition> defines an assertion.

   Note: Subclause 10.8, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked.

2) The assertion is not satisfied if and only if the result of evaluating the <search condition> is false.

3) An assertion descriptor is created that describes the assertion being defined. The name included in the assertion descriptor is <constraint name>.

   The assertion descriptor includes an indication of whether the constraint is deferrable or not deferrable and whether the initial constraint mode is *deferred* or *immediate*.

   The assertion descriptor includes the <search condition> of the assertion.

4) If the character representation of the <search condition> cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning—search condition too long for information schema*.

5) If an <assertion definition> specifies an <immediate assertion trigger>, then

   Case:

   a) If INSERT is specified, the evaluation of <triggered assertion>s takes place at the end of an <insert statement> whose subject table is *T1*.

   b) If DELETE is specified, the evaluation of <triggered assertion>s takes place at the end of each <delete statement: positioned> or <delete statement: searched> whose subject table is *T1*.

   c) If UPDATE is specified without an <assertion column list>, the evaluation of <triggered assertion>s takes place at the end of each <update statement: positioned> or <update statement: searched> whose subject table is *T1*.

   d) If UPDATE is specified with an <assertion column list>, the evaluation of <triggered assertion>s takes place at the end of each <update statement: positioned> or <update statement: searched> whose subject table is *T1* and whose <set clause list> identifies any of the column names in the <assertion column list>.

6) An SQL-statement that triggers an <immediate assertion trigger> is referred to as the triggering statement for the assertion.

7) If <deferred assertion trigger> is specified, then the evaluation of <triggered assertion>s takes place immediately before the execution of a <commit statement>.

8) Case:

   a) If FOR is specified, the <triggered assertion> is evaluated for each row of *T2*. Any outer reference to a column of *T2* is considered to be a reference to the value of the column in that row of *T2*.

   Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

   b) If FOR is not specified, then the <triggered assertion> is evaluated once.

9) The inclusion of EACH ROW OF has no effect on the evaluation of the assertion.

10) If the constraint mode of the assertion in an SQL-transaction is *deferred*, then the checking of an assertion triggered by an <immediate assertion trigger> is deferred.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <assertion definition> shall contain no <assertion trigger>.

   b) A <triggered assertion> shall not specify FOR.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <assertion definition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.40 <drop assertion statement>

Function

Destroy an assertion.

Format

```
<drop assertion statement> ::=
       DROP ASSERTION <constraint name>
```

Syntax Rules

1) Let *A* be the assertion identified by <constraint name> and let *AN* be the name of *A*.

2) The schema identified by the explicit or implicit schema name of *AN* shall include the descriptor of *A*.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the assertion identified by *AN*.

General Rules

1) The descriptor of *A* is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <drop assertion statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.41 <trigger definition>

Function

Define triggered SQL-statements.

Format

```
<trigger definition> ::=
```

ANSI Only--SQL3

```
       [ CREATE ] TRIGGER [ <trigger name> ]
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
       CREATE TRIGGER <trigger name>

         <trigger action time> <trigger event>
```

ANSI Only--SQL3

```
           [ ON <table name> ]
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
           ON <table name>
```

ANSI Only--SQL3

```
           [ REFERENCING <old or new values alias list> ]
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
           [ REFERENCING
               { OLD [ AS ] <old values correlation name>
                   [ NEW [ AS ] <new values correlation name> ] }
             | { NEW [ AS ] <new values correlation name>
                   [ OLD [ AS ] <old values correlation name> ] } ]

         <triggered action>...

<trigger action time> ::=
         BEFORE
       | AFTER
```

ANSI Only--SQL3

```
       | INSTEAD OF
```

```
<trigger event> ::=
         INSERT
       | DELETE
       | UPDATE [ OF <trigger column list> ]

<trigger column list> ::= <column name list>

<triggered action> ::=
       [ WHEN <left paren> <search condition> <right paren> ]
```

ANSI Only--SQL3

```
         <triggered SQL statement>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
         <left paren> <triggered SQL statement list> <right paren>

            [ FOR EACH { ROW | STATEMENT } ]
```

ANSI Only--SQL3

```
<triggered SQL statement> ::=
       <SQL procedure statement> <semicolon>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
<triggered SQL statement list> ::=
       <triggered SQL statement> [ { <comma> <triggered SQL statement> }... ]

<triggered SQL statement> ::=
         <delete statement: searched>
       | <update statement: searched>
       | <insert statement>
```

ANSI Only--SQL3

```
<old or new values alias list> ::=
       <old or new values alias>...

<old or new values alias> ::=
         <OLD [ AS ] <old values correlation name>
       | NEW [ AS ] <new values correlation name>
       | <OLD_TABLE [ AS ] <old values table alias>
       | <NEW_TABLE [ AS ] <new values table alias>

<old values table alias> ::= <identifier>

<new values table alias> ::= <identifier>

<old values correlation name> ::= <correlation name>
```

```
<new values correlation name> ::= <correlation name>
```

Syntax Rules

1) Case:

   a) If a <trigger definition> is contained in a <schema definition> and if the <trigger name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

   b) If a <trigger definition> is contained in a <module> and if the <trigger name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the <module>.

ANSI Only–SQL3

2) Case:

   a) If the <trigger definition> is contained in a <query expression>, then none of CREATE, <trigger name>, or ON <table name> shall be specified. The <trigger definition> is effectively applied to the <derived table> defined by the <query expression>.

   b) Otherwise, CREATE, <trigger name>, and ON <table name> shall all be specified.

****Editor's Note****

The previous Rule makes it permissible to define a trigger on a <query expression>, which defines an unnamed virtual table. However, succeeding Rules require that the table on which the trigger is defined have a name. This has been identified as a Possible Problem. See Possible Problem **269** in the Editor's Notes.

3) Let *T* be the <table name> of a <trigger definition>.

ISO Only—caused by ANSI changes not yet considered by ISO

T shall identify a base table.

The table identified by *T* is referred to as the *subject table* of the <trigger definition>.

4) If a <trigger column list> is specified, then the <column name>s of the <trigger column list> shall identify columns of the table identified by *T*. Otherwise there is an implied <trigger column list> that identifies all the columns defined in the <table definition> of *T*.

ANSI Only–SQL3

5) The <triggered SQL statement> shall not generally contain an <SQL schema statement>, an <SQL transaction statement>, an <SQL connection statement>, an <SQL session statement>, or an <SQL dynamic statement>.

6) No <triggered SQL statement> shall generally contain any <SQL data statement> other than <delete statement: searched>, <update statement: searched>, and <insert statement>.

7) No <column name> shall appear more than once in the <triggered column list>.

ANSI Only–SQL3

8) If REFERENCING is specified, then:

   a) OLD, NEW, OLD_TABLE, and NEW_TABLE shall be specified at most once each within the <old or new values alias list>.

   b) Case:

      i) If <trigger event> specified INSERT and <trigger action time> specifies BEFORE, then neither OLD nor OLD_TABLE shall be specified.

      ii) If <trigger event> specified DELETE and <trigger action time> specifies AFTER, then neither NEW nor NEW_TABLE shall be specified.

   c) No two of <old values correlation name>, <old values table alias>, <new values correlation name>, <new values table alias>, and *T*, treated as a <character string literal>, shall compare equal.

   d) Let *TN* be the <table name> specified in the <delete statement: searched>, <udpate statement: searched>, or <insert statement> that is a <triggered SQL statement>. No two of <old values correlation name>, <old values table alias>, <new values correlation name>, <new values table alias>, and *TN*, treated as a <character string literal>, shall compare equal.

   e) The scope of the <old values correlation name>, <old values table alias>, <new values correlation name>, <new values table alias>, and *T* is the entire <trigger definition>.

ISO Only—caused by ANSI changes not yet considered by ISO

9) If REFERENCING is specified and the <trigger event> specifies DELETE or INSERT, then

   Case:

   a) If <trigger event> specifies DELETE and <trigger action time> specifies AFTER, then
   ANSI <new values name>
   ISO <new values correlation name>
   shall not be specified.

   b) If <trigger event> specifies INSERT and <trigger action time> specifies BEFORE, then
   ANSI <old values name>
   ISO <old values correlation name>
   shall not be specified.

10) If REFERENCING is specified and <trigger event> specifies UPDATE, then:

a) The
[ANSI] <old values name>,
[ISO] <old values correlation name>,
the
[ANSI] <new values name>,
[ISO] <new values correlation name>,
and *T* shall be different.

b) Let *TN* be the <table name> specified in the <delete statement: searched>, <update statement: searched>, or <insert statement> that is a <triggered SQL statement>. *TN* shall not be the same as the
[ANSI] <old values name> or the <new values name>.
[ISO] <old values correlation name> or the <new values correlation name>.

c) The scope of the
[ANSI] <old values name>, the <new values name>,
[ISO] <old values correlation name>, the <new values correlation name>,
and *T* is the entire <trigger definition>.

11) The <search condition> and <triggered SQL statement>s shall not contain a <value specification> that specifies a
[ANSI] parameter reference or a <dynamic parameter specification>.
[ISO] <target specification>.

ANSI Only–SQL3

12) If neither FOR EACH ROW nor FOR EACH STATEMENT is specified, then FOR EACH STATEMENT is implicit.

13) If <old values table alias> or <new values table alias> is specified, then:

a) FOR EACH STATEMENT shall be specified.

b) The <identifier> contained in <old values table alias> or <new values table alias> shall be referenced only in a <triggered action> containing FOR EACH STATEMENT.

14) If an <old values table alias> or <new values table alias> is referenced in a <triggered action>, then it shall refer to a read-only table.

15) If <old values correlation name> or <new values correlation name> is specified, then:

a) FOR EACH ROW shall be specified.

b) The <identifier> contained in <old values table alias> or <new values table alias> shall be referenced only in a <triggered action> containing FOR EACH ROW.

16) If a <triggered action> contains an outer reference to *T*, then the <trigger definition> shall specify FOR EACH ROW.

ISO Only—caused by ANSI changes not yet considered by ISO

17) If the <triggered action> does not specify FOR EACH ROW, then <triggered action> shall not contain an outer reference to *T*.

Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

18) If BEFORE INSERT or AFTER DELETE is specified and a REFERENCES clause is not specified, then the <triggered action> shall not specify an outer reference to *T*.

19) Let *D1* and *D2* be two <trigger definition>s with the same subject table *T* and the same <trigger action time>.

Case:

a) If the <trigger event> of *D1* specifies INSERT or DELETE, then *D2* shall not specify an identical <trigger event>.

b) If the <trigger event> of *D1* specifies UPDATE, then *D2* shall not specify a <trigger column list> that references any of the <column name>s in the <trigger column list> of *D1*.

20) If the <trigger event> specifies DELETE, then *T* shall not be

Case:

a) the referencing table in any <referential constraint definition> that specifies ON DELETE CASCADE.

b) the referenced table in any <referential constraint definition> that specifies PENDANT.

21) If the <trigger event> specifies UPDATE, then let C_i be the *i*-th <column name> in the <trigger column list>. *T* shall not be the referencing table in any <referential constraint definition> that specifies ON UPDATE CASCADE, ON UPDATE SET NULL, ON UPDATE SET DEFAULT, ON DELETE SET NULL, or ON DELETE SET DEFAULT and contains a <reference column list> that includes C_i.

ANSI Only–SQL3

22) The schema identified by the explicit or implicit schema name of the <trigger name> shall not include a trigger descriptor whose trigger name is <trigger name>.

ISO Only—caused by ANSI changes not yet considered by ISO

23) The <qualified identifier> of <trigger name> shall be different from the <qualified identifier> of the <trigger name> of any other <trigger definition> in the same schema.

---

24) The *trigger action graph* is a directed graph whose nodes are defined as follows:

a) Let *T* be the <table name> that is the subject table name in a <trigger definition> and let *E* be the <trigger event> in that <trigger definition>. TABLE(*T*) is a node of the graph.

b) If *E* specifies DELETE, then DELETE(*T*) is a node of the graph.

c) If *E* specifies INSERT, then INSERT(*T*) is a node of the graph.

d) If *E* specifies UPDATE, then UPDATE(*T*) and UPDATE(*T.C*) are nodes on the graph for each <column name> *C* appearing in the explicit or implicit <trigger column list>.

e) Let T_s be any table that appears as the <table reference> of any <triggered SQL statement> or let T_s be any leaf generally underlying table of any view V_s that appears as the <table reference> of any <triggered SQL statement>.

f) If the <triggered SQL statement> is an <insert statement>, then INSERT(T_s) and TABLE(T_s) are nodes of the graph.

g) If the <triggered SQL statement> is a <delete statement: searched>, then DELETE(T_s) and TABLE(T_s) are nodes of the graph.

h) If the <triggered SQL statement> is an <update statement: searched>, then UPDATE(T_s. C_s), UPDATE(T_s) and TABLE(T_s) are nodes of the graph for each <column name> C_s of T_s that appears explicitly or implicitly as an <object column> in the <triggered SQL statement>.

25) The arcs of the trigger action graph are defined as follows:

a) If *E* specifies DELETE, then there is an arc from DELETE(*T*) to TABLE(*T*) and

Case:

i) If the
ANSI <triggered SQL statement>
ISO <triggered SQL statement list>
contains an <insert statement>, then there is an arc from DELETE(*T*) to INSERT(T_s) and an arc from INSERT(T_s) to TABLE(T_s).

ii) If the
ANSI <triggered SQL statement>
ISO <triggered SQL statement list>
contains a <delete statement: searched>, then there is an arc from DELETE(*T*) to DELETE(T_s) and an arc from DELETE(T_s) to TABLE(T_s).

iii) If the
ANSI <triggered SQL statement>
ISO <triggered SQL statement list>
contains an <update statement: searched> with an <object column> that references a <column name> C_s of T_s, then there is an arc from DELETE(*T*) to UPDATE($T_s.C_s$), an arc from UPDATE($T_s.C_s$) to UPDATE(T_s), and an arc from UPDATE(T_s) to TABLE(T_s).

b) If *E* specifies INSERT, then there is an arc from INSERT(*T*) to TABLE(*T*) and

Case:

i) If the
[ANSI] <triggered SQL statement>
[ISO] <triggered SQL statement list>
contains an <insert statement>, then there is an arc from INSERT(*T*) to INSERT(T_s)
and an arc from INSERT(T_s) to TABLE(T_s).

ii) If the
[ANSI] <triggered SQL statement>
[ISO] <triggered SQL statement list>
contains a <delete statement: searched>, then there is an arc from INSERT(*T*) to DELETE(T_s) and an arc from DELETE(T_s) to TABLE(T_s).

iii) If the
[ANSI] <triggered SQL statement>
[ISO] <triggered SQL statement list>
contains an <update statement: searched> with an <object column> that references a <column name> C_s of T_s, then there is an arc from INSERT(*T*) to UPDATE(T_s.C_s), an arc from UPDATE(T_s.C_s) to UPDATE(T_s) and an arc from UPDATE(T_s) to TABLE(T_s).

c) If *E* specifies UPDATE and references a <column name> *C* in a <trigger column list> either explicitly or implicitly, then there is an arc from UPDATE(*T*.*C*) to UPDATE(*T*), an arc from UPDATE(*T*) to TABLE(*T*) and

Case:

i) If the
[ANSI] <triggered SQL statement>
[ISO] <triggered SQL statement list>
contains an <insert statement>, then there is an arc from UPDATE(*T*) to INSERT(T_s)
and an arc from INSERT(T_s) to TABLE(T_s).

ii) If the
[ANSI] <triggered SQL statement>
[ISO] <triggered SQL statement list>
contains a <delete statement: searched>, then there is an arc from UPDATE(*T*) to DELETE(T_s) and an arc from DELETE(T_s) to TABLE(T_s).

iii) If the
[ANSI] <triggered SQL statement>
[ISO] <triggered SQL statement list>
contains an <update statement: searched> with an <object column> that references a <column name> C_s of T_s, then there is an arc from UPDATE(*T*) to UPDATE(T_s.C_s), an arc from UPDATE(T_s.C_s) to UPDATE(T_s), and an arc from UPDATE(T_s) to TABLE(T_s).

26) Let *G* be the trigger action graph and let *T* be the <table name> of the subject table in a <trigger definition> *D*. The *TD_trigger definition* subgraph of *G* is defined as follows:

a) Case:

i) If the <trigger event> of *D* specifies UPDATE, then for any column *C* of table *T*, if UPDATE(*T*.*C*) is a node of *G*, then that node is a node of the subgraph.

ii) If the <trigger event> of *D* specifies DELETE, then DELETE(*T*) is a node of the subgraph.

iii) If the <trigger event> of *D* specifies INSERT, then INSERT(*T*) is a node of the subgraph.

b) The following rule is applied recursively:

If *N* is a node of the subgraph and *N1* is a node of *G* and there exists an arc from *N* to *N1* in *G*, then the node denoted by *N1* is a node of the subgraph and the arc from *N* to *N1* is an arc of the subgraph.

c) A TD_trigger definition subgraph of *G* is defined to be *consistent* if and only if for each node in the subgraph other than those of the form UPDATE(*X*) for some table *X*, the node has at most 1 incoming arc.

27) The TD_trigger definition subgraph of the trigger action graph shall be consistent.

- 1 Rule deleted (moved to ARs).

Access Rules

1) If a <trigger definition> is contained in a <module>, then the <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <trigger name> of the <trigger definition>.

2) Let *TN* be any <table name> generally contained in a <search condition> contained in the <trigger definition> or generally contained in a <subquery> in a <triggered SQL statement> contained in the <trigger definition>. The applicable privileges for *TN* shall include SELECT.

3) Let *T* be the subject table of the <trigger definition>. The applicable privileges for *T* shall include TRIGGER.

4) Let *RT* be the table identified by the <table reference> of a <triggered SQL statement> *S*. The applicable privileges for *RT* shall include:

Case:

a) If *S* is an <insert statement>, then

Case:

i) If *S* contains an <insert column list>: INSERT for each <column name> in the <insert column list>.

ii) Otherwise: INSERT.

b) If *S* is an <update statement: searched>: UPDATE for each <object column>.

c) If *S* is a <delete statement: searched>: DELETE.

General Rules

1) A <trigger definition> defines a trigger.

ANSI Only–SQL3

A trigger descriptor is created as follows:

a) The trigger name included in the trigger descriptor is <trigger name>.

b) The subject table included in the trigger descriptor is <table name>.

c) The trigger action time included in the trigger descriptor is <trigger action time>.

d) The trigger event included in the trigger descriptor is <trigger event>.

e) If the <trigger definition> specifies an <old values correlation name>, then the old values correlation name included in the trigger descriptor is that <old values correlation name>.

f) If the <trigger definition> specifies an <new values correlation name>, then the new values correlation name included in the trigger descriptor is that <new values correlation name>.

g) The trigger actions included in the trigger descriptor are each of the specified <triggered action>s.

h) If the <trigger event> specifies UPDATE, then the trigger column list included in the trigger descriptor is the implicit or explicit <trigger column list>.

****Editor's Note****
Paper X3H2-92-219/CBR-055 noted that the preceding General Rule should be modified to create a trigger descriptor.

ANSI Only–SQL3

2) If the <trigger definition> *TD* contains a <trigger action time> that specifies INSTEAD OF and a <trigger event> that specifies UPDATE, then:

a) For each column *C* whose <column name> is in the <trigger column list> of *TD*, indicate in the descriptor for *C* that *C* is updatable;

b) If the <table name> of *TD* identifies a derived table *V* and if *V* will be an updatable derived table after creation of this trigger, then indicate that *V* is an updatable table in the descriptor for *V*.

3) Let *VN* be the <table name> specified in a <view definition> in which *T* is a leaf generally underlying table. Let *S*, a triggering SQL-statement, be any <SQL data change statement> that specifies a <table reference> of *T* or *VN*. If *S* specifies *T*, then let *RS* be the empty or non-empty

set of rows from *T* identified by *S*. If *S* specifies *VN*, then let *RS* be the empty or non-empty set of rows from *T* affected by *S*.

A <trigger definition> specifies the <triggered action>s that are effectively executed whenever *S* is executed.

Case:

a) If BEFORE INSERT is specified, then the <triggered action>s are executed before the insert of *RS* into *T*. The rows *RS* are not visible to the <search condition> or <triggered SQL statement>s.

b) If AFTER INSERT is specified, then the <triggered action>s are executed after the insert of *RS* into *T*. The rows *RS* are visible to the <search condition> or <triggered SQL statement>s.

ANSI Only–SQL3

c) If INSTEAD OF INSERT is specified, then the triggered action is executed instead of the insertion of *RS* into *T*. If <new values correlation name> or <new values table alias> is specified, then the rows *RS* are visible to the <triggered SQL statement>s.

d) If BEFORE DELETE is specified, then the <triggered action>s are executed before the deletion of *RS* from *T*. The rows *RS* are visible to the <search condition> or <triggered SQL statement>s.

e) If AFTER DELETE is specified, then the <triggered action>s are executed after the deletion of *RS* from *T*. The rows *RS* are not visible to the <search condition> or <triggered SQL statement>s.

ANSI Only–SQL3

f) If INSTEAD OF DELETE is specified, then the triggered action is executed instead of the deletion of *RS* from *T*. If <old values correlation name> or <old values table alias> is specified, then the rows *RS* are visible to the <triggered SQL statement>s.

g) If BEFORE UPDATE is specified, then the <triggered action>s are effectively executed before the update of *RS* in *T*. The values in *RS* that existed prior to the execution of *S* are visible to the <search condition> or <triggered SQL statement>s. If <new values correlation name> is not specified, then the updated values in *RS* are not visible to the <search condition> or <triggered SQL statement>s. If <new values correlation name> is specified, then the updated values in *RS* are accessible to the <search condition> or <triggered SQL statement>s by means of an outer reference that is qualified by the <new values correlation name>.

h) If AFTER UPDATE is specified, then the <triggered action>s are effectively executed after the update of *RS* in *T*. The updated values in *RS* are visible to the <search condition> or <triggered SQL statement>s. If <old values correlation name> is not specified, then the values in *RS* that existed prior to the execution of *S* are not visible to the <search condition> or <triggered SQL statement>s. If <old values correlation name> is specified, then the values in *RS* that existed prior to the execution of *S* are accessible to the <search condition> or <triggered SQL statement>s by means of an outer reference that is qualified by the <old values correlation name>.

ANSI Only–SQL3

i) If INSTEAD OF UPDATE is specified, then the triggered action is executed instead of the update of *RS* in *T*. If <new values correlation name> or <new values table alias> is specified, then the rows *RS* are visible to the <triggered SQL statement>s. If <old values correlation name> or <old values table alias> is specified, then the values in *RS* that existed prior to the execution of *S* are visible to the <triggered SQL statement>s.

4) If a <trigger definition> contains more than one <triggered action>, then they are effectively executed in the order in which they are specified in the <trigger definition>.

5) If a <triggered action> specifies FOR EACH ROW then the <triggered action> is executed once for each row of *RS*. Otherwise the <triggered action> is executed once.

6) Let *C* be a <column name> of *T* referenced as an outer reference in a <trigger definition>.
Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

7) If the outer reference to *C* is contained in a <trigger definition> containing a <trigger event> that specifies UPDATE, then let *SU* be a triggering SQL-statement that is an <update statement: positioned> or <update statement: searched> that references *T*.
Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

Case:

a) If the reference to *C* is qualified by <new values correlation name>, then the reference to *C* is to the value that will be present in the column identified by *C* in some row of *RS* after the update of that row by *SU*.

b) If the reference to *C* is qualified by <old values correlation name>, then the reference to *C* is to the value that was present in the column identified by *C* in some row of *RS* before the update of that row by *SU*.

c) If the reference to *C* is not qualified by either <new values correlation name> or <old values correlation name>, then

Case:

i) If the <trigger action time> specifies AFTER, then the reference to *C* is to the value that will be present in the column identified by *C* in some row of *RS* after the update of that row by *SU*.

ii) If the <trigger action time> specifies BEFORE, then the reference to *C* is to the value that was present in the column identified by *C* in some row of *RS* before the update of that row by *SU*.

8) If the outer reference to *C* is contained in a <trigger definition> containing a <trigger event> that specifies DELETE or INSERT, then the reference to *C* is to the value of *C* in some row of *RS*.

Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

ANSI Only–SQL3

9) If a reference to an <old values table alias> is contained in a <triggered action>, then it refers to the whole collection of rows *RS* before any effect of the triggering SQL-statement.

10) If a reference to a <new values table alias> is contained in a <triggered action>, then:

Case:

a) If the <trigger event> is INSERT, then the reference is to the whole collection of candidate rows *RS* to be inserted, that is, with defaults already applied.

b) If the <trigger event> is UPDATE, then the reference is to the whole collection of rows *RS* that would have resulted from updating the original collection *RS* as though it were a base table.

11) Case:

a) If a <triggered action> contains a <search condition>, then

Case:

i) If the <search condition> is true, then the <triggered SQL statement>s

ISO Only—caused by ANSI changes not yet considered by ISO

in the <triggered SQL statement list>

are executed in the order in which they are specified.

ii) If the <search condition> is not true, then the <triggered SQL statement>s

ISO Only—caused by ANSI changes not yet considered by ISO

in the <triggered SQL statement list>

are not executed.

b) If a <triggered action> does not contain a <search condition>, then the <triggered SQL statement>s

ISO Only—caused by ANSI changes not yet considered by ISO

in the <triggered SQL statement list>

are executed in the order in which they are specified.

12) Let *ID* be the <table name> that is the subject of any <triggered SQL statement> that is an <insert statement>. Let *DT* be the <table name> that is the subject of any <triggered SQL statement> that is a <delete statement: searched>. Let *UT* be the <table name> that is the subject of any <triggered SQL statement> that is an <update statement: searched>. A <triggered action> is *cascaded* when

Case:

a) *ID* appears as the <table name> in a <trigger definition> that has a <trigger event> that specifies INSERT.

b) *DT* appears as the <table name> in a <trigger definition> that has a <trigger event> that specifies DELETE.

c) *UT* appears as the <table name> in a <trigger definition> that has a <trigger event> that specifies UPDATE.

13) If the execution of a <triggered SQL statement> inserts rows, deletes rows, or updates rows of a table *TX*, then any cascaded triggered actions for *TX* are activated by the <triggered SQL statement>s as defined by the <trigger definition>s containing the <table name> *TX*.

14) If the execution of the <triggered SQL statement> for two rows R_1 and R_2 of *RS* result in the update of a column *C* for the same row R_x in some table and if the <triggered SQL statement> for R_1 and R_2 result in different values for *C*, then an exception condition is raised: *triggered data change violation*.

15) If the cursor mode of the current SQL-transaction is set to *cascade off*, then all <triggered SQL statement>s are effectively deferred until implicitly enacted by execution of a <commit statement> or a <close statement>.

16) If the execution of a <triggered SQL statement> is not successful, then an exception condition is raised: *triggered action exception*. The exception information associated with the <triggered SQL statement> is entered into the diagnostics area in a location other than the location corresponding to condition number 1.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) Conforming Full SQL language shall contain no <trigger definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.42 <drop trigger statement>

Function

Destroy a trigger.

Format

```
<drop trigger statement> ::= DROP TRIGGER <trigger name>
```

Syntax Rules

1) Let *TR* be the trigger identified by the <trigger name>.

Access Rules

1) The <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of *TR*.

General Rules

ANSI Only–SQL3

1) If *TR* was specified with a <trigger action time> that specifies INSTEAD OF, and a <trigger event> that specifies UPDATE, then:

   a) For each column *C* whose <column name> was specified in the <trigger column list> for *TR*, indicate in the descriptor for *C* whether *C* is still updatable or is now read-only;

   b) If the <table name> that was specified for *TR* identifies a derived table *V*, then indicate in the descriptor for *V* whether *V* is now updatable or read-only.

2) The identified trigger and its descriptor are destroyed.

ISO Only—caused by ANSI changes not yet considered by ISO

3) The trigger is dropped from the schema and the trigger descriptor is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop trigger statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.43 <drop routine statement>

Function

Destroy an SQL routine.

Format

```
<drop routine statement> ::=
```

ANSI Only--SQL3

```
      DROP ROUTINE { SPECIFIC <specific name> | <member name> } <drop behavior>

<member name> ::= <routine name> [ <data type list> ]
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
      DROP ROUTINE <routine name>
          [ <data type list> ] <drop behavior>


<data type list> ::=
      <left paren> <data type> [ { <comma> <data type> }... ] <right paren>
```

Syntax Rules

ANSI Only–SQL3

1) If <specific name> is specified, then:

   a) Let *SR* be the routine identified by the <specific name>.

   b) Let *RF* be the routine family identified by the <routine name> of *SR*.

2) If <routine name is specified, then:

   a) Let *RF* be the routine family identified by the <routine name> *SRN*.

   b) *RF* shall identify a routine family in the catalog identified by the explicit or implicit <catalog name>.

   c) If *RF* identifies a routine family with more than one member, then a <data type list> shall be specified.

   d) If no <data type list> is specified, then the single member *SR* of the routine family is identified.

e) If a <data type list> is specified, then the *i*-th <data type> shall be identical to the *i*-th <data type> in the <parameter declaration list> of a function in *RF*. This is the identified function *SR*.

f) *SR* shall not have been created as part of an <abstract data type definition>.

ISO Only—caused by ANSI changes not yet considered by ISO

3) Let *RF* be the routine family identified by the <routine name> *SRN*.

4) *RF* shall identify a routine family in the catalog identified by the explicit or implicit <catalog name>.

5) If *RF* identifies a routine family with more than one member, then a <data type list> shall be specified.

6) If no <data type list> is specified, then the single member *SR* of the routine family is identified.

7) If a <data type list> is specified, then the *i*-th <data type> shall be identical to the *i*-th <data type> in the <parameter declaration list> of a function in *RF*. This is the identified function *SR*.

8) *SR* shall not have been created as part of an <abstract data type definition>.

9) If RESTRICT is specified, then *SR* shall not be referenced in the <query expression> of any view descriptor, the <search condition> of any constraint descriptor, in any trigger descriptor, domain descriptor, or abstract data type descriptor.

Access Rules

1) The <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <routine name>.

General Rules

1) If *RF* identifies a routine family with exactly one member, then for every privilege descriptor with a grantor of ''_SYSTEM'' that specifies some FUNCTION PRIVILEGES on *SR*, the following <revoke statement> is effectively executed as though the <authorization identifier> of the current SQL-session were ''_SYSTEM'' and without further Access Rule checking:

REVOKE FUNCTION PRIVILEGES ON *EN* FROM *GRANTEE* CASCADE

where *GRANTEE* is the grantee in the privilege descriptor.

ANSI Only–SQL3

2) The identified routine and its descriptor are destroyed.

ISO Only—caused by ANSI changes not yet considered by ISO

3) The routine associated with the <routine> is destroyed.

4) The <routine> is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop routine statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.44 <abstract data type definition>

Function

Define an abstract data type.

Format

```
<abstract data type definition> ::=
         <distinct type definition>
       | <explicit abstract data type definition>

<explicit abstract data type definition> ::=
       CREATE TYPE <abstract data type name>
         <abstract data type body>
```

Syntax Rules

1) Case:

   a) If an <explicit abstract data type definition> is contained in a <schema definition>, and if the <explicit abstract data type definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

   b) If an <explicit abstract data type definition> is contained in a <module>, and if the <explicit abstract data type definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <module>.

2) The <qualified identifier> of <abstract data type name> shall be different from the <qualified identifier> of any other <abstract data type name> of an <explicit abstract data type definition> in the same schema and shall be different from the <qualified identifier> of any <domain name> in the same schema.

Access Rules

None.

General Rules

1) An <explicit abstract data type definition> defines an abstract data type and the functions that are associated with it.

2) A value of an abstract data type consists of a value of the stored attributes specified by the ordered sequence of column descriptors in the abstract data type descriptor. The values corresponding to the actual base types of such an abstract data type value are the *actual base items* of that value.

3) A privilege descriptor is created that defines the USAGE privilege for the <abstract data type name> to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <abstract data type name>. This privilege is grantable. The grantor for this privilege descriptor is set to the special grantor value ''_SYSTEM''.

4) A privilege descriptor is created granting the UNDER privilege on the abstract data type to the owner; the grantor is the special grantor value ''_SYSTEM''. If and only if the UNDER privilege is held on all supertypes of the abstract data type WITH GRANT OPTION will be privilege descriptor be specified WITH GRANT OPTION.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <abstract data type definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.45 <abstract data type body>

Function

Specify an <abstract data type body>.

Format

```
<abstract data type body> ::=
       [ <oid options> ]
       [ <subtype clause> ]
       [ [ <constant or updatable> ] [ <member list> ] ]

<subtype clause> ::=
       UNDER <supertype clause> [ { <comma> <supertype clause> }... ]

<supertype clause> ::=
       <abstract data type name> [ <component renaming clause> ]

<component renaming clause> ::=
       WITH <left paren> <component renaming element>
            [ { , <component renaming element> }... ] <right paren>

<component renaming element> ::=
       <supertype component name> AS <subtype component name>

<supertype component name> ::= <component name>

<subtype component name> ::= <component name>

<member list> ::=
       <left paren> <member> [ { <comma> <member> }... ] <right paren>

<member> ::=
         <column definition>
       | <routine declaration>
       | <operator name list>
       | <equals clause>
       | <less-than clause>
       | <cast clause>
```

ANSI Only--SQL3

```
       | <table constraint definition>
```

```
<operator name list> ::=
       OPERATORS <operator name>...

<operator name> ::= <routine name>

<equals clause> ::=
       EQUALS <equals function specification> <semicolon>

<less-than clause> ::=
       LESS THAN <less-than function specification> <semicolon>
```

```
<cast clause> ::=
       CAST <left paren> <operand data type> AS <result data type>
            WITH <cast function> <right paren> <semicolon>

<cast function> ::=
       <routine name>

<equals function specification> ::=
         <routine name>
       | DEFAULT
       | REF

<less-than function specification> ::=
         <routine name>
       | NONE

<routine declaration> ::=
       [ <encapsulation level> ] <routine>

<operand data type> ::= <data type>

<result data type> ::= <data type>
```

Syntax Rules

1) Case:

   a) If <abstract data type body> immediately contains CONSTANT, then CONSTANT is implicit for each <stored column> contained in the <member list>.

   b) If <abstract data type body> immediately contains UPDATABLE, then there shall exist at least one <stored column> contained in the <member list> that does not specify CONSTANT.

2) <operator name list> shall be specified no more than once.

3) Let *STD* be the <abstract data type definition> and let *ST* be the abstract data type defined by *STD*.

4) If <subtype clause> is not specified, then <member list> shall be specified.

5) If <member list> is specified, then:

   a) Let *n* be the number of <column definition>s contained in <member list>. Let *originally-defined representation* be the <column definition> contained in <member list>.

   b) Let *originally-defined component* be a component defined by a <column definition> contained in <member list>. Let C_{rk} be the *k*-th originally-defined component.

ANSI Only–SQL3

---

   c) Each <table constraint definition> contained in the <element list> shall contain <check constraint definition>.

---

6) If <subtype clause> is specified, then

Case:

a) If the <subtype clause> contains any <abstract data type name> whose abstract data type descriptor specifies WITH OID VISIBLE, then

Case:

i) If <oid options> is specified, then it shall be WITH OID VISIBLE.

ii) Otherwise, WITH OID VISIBLE is implicit.

b) If, for every <abstract data type name> in the <subtype clause>, the associated abstract data type descriptor specifies WITHOUT OID and <oid options> is not specified, then WITHOUT OID is implicit.

7) If <oid options> is not specified, then WITH OID NOT VISIBLE is implicit.

8) If <subtype clause> is specified, then:

a) No <abstract data type name> shall be simply contained more than once in <subtype clause>.

b) Let *m* be the number of <supertype clause>s contained in the <subtype clause>. Let *k(i)* be the number of <column definition>s in the implicit tuple definition of the <abstract data type definition> that defined the abstract data type identified by the <abstract data type name> simply contained in the *i*-th <supertype clause>. Let C_{ij} be the *j*-th <component definition> of the <abstract data type definition> that defined the abstract data type identified by the <abstract data type name> simply contained in the *i*-th <supertype clause>.

c) The *inherited representation* of *STD* is:

$$C_{11}, \cdots, C_{1\,k(1)}, \cdots, C_{i1}, \cdots, C_{i\,k(i)}, \cdots, C_{m1}, \cdots, C_{m\,k(m)}$$

d) Let *inherited component* be a component of the inherited representation of *STD*.

e) If <component renaming clause> is specified whose <supertype component name> identifies some inherited component, then the effective <column name> in the inherited representation of *STD* is the <subtype component name> specified in the <component renaming clause>.

9) No <column name> immediate contained in a <column definition> of the <abstract data type body> nor any <component name> in a <component renaming clause> shall be OID.

10) Let *ADTN* be the <abstract data type name> immediately contained in *STD*. If <oid options> specifies WITH OID, then a <column definition> that specifies ''OID REF(*ADTN*) NOT NULL'' is implicit. Any inherited component with <column name> OID is deleted from the inherited representation of *STD*.

****Editor's Note****

There is a general problem in this section in that where <abstract data type body> is contained in a <type template definition> and not in an <abstract data type definition>, *STD* is not defined. When the fix to cover type templates is made, it should probably handle this rule by generating merely OID REF NOT NULL. Then <generated type reference> in Subclause 10.4, "<generated type reference>", should tighten REF to REF(*IDN*) whenever it generates a specific type name *IDN* from the template.

11) Case:

a) If <member list> is not specified, then an implicit <member list> *TUP* is constructed for *STD* that is the inherited representation of *STD*.

b) Otherwise, an implicit <member list> *TUP* is constructed for *STD* that is the inherited representation of *STD* followed by the explicit <member list> of *STD*:

$$C_{11}, \cdots, C_{1\,k(1)}, \cdots, C_{i1}, \cdots, C_{i\,k(i)}, \cdots, C_{m1}, \cdots, C_{m\,k(m)}, C_{r1}, \cdots, C_{rn}$$

12) Let *same component* mean a component of *TUP* that appears in more than one direct supertype of *ST* that is inherited by each of those direct supertypes from the same component of a single higher-level supertype. (The <component name> of the same component in a direct supertype of *ST* need not be the same as in other direct supertypes of *ST*.)

13) If some component *SC* is the same component in more than one direct supertype of *ST*, then all occurrences of *SC* except the first occurrence are dropped from *TUP*.

14) The effective <member list> *DTR* of *STD*, replacing any explicit <member list> of *STD*, is:

TUP;

15) In *DTR*, no <column name> shall appear more than once.

16) *ST* shall be a member of exactly one supertype family.

17) An <abstract data type definition> *ADT* is *directly based on* a data type *DT* if *DT* is contained in a <column definition> of *ADT* with no intervening <object identifier type>. An <abstract data type definition> *ADT* is *based on* a data type *DT* if *ADT* is directly based on *DT* or if *ADT* is directly based on an abstract data type ADT2 that is based on *DT*.

18) The *actual base type* of a predefined type *PDT* is *PDT*. The *actual base types* of an abstract data type *ADT* are the actual base types of the <data type>s of the components of *ADT*, in the order that those components are specified in the <abstract data type definition> of *ADT*.

19) An abstract data type shall not be based on itself.

20) If no <equals function specification> is present, then DEFAULT is implicit.

21) If no <less than function specification> is present, then NONE is implicit.

22) At most one <equals clause> and one <less than clause> shall be contained in the <member list>.

23) Case:

a) If <equals function specification> specifies DEFAULT, then:

i) Let *ADTN* be the <abstract data type name>.

ii) Let *C1*, . . . , *Cn* be the components of the representation of the abstract data type, excluding any component with <column name> OID..

iii) The <abstract data type definition> effectively contains an implicit <routine> that specifies:

```
FUNCTION ADT_EQUALS ( ADT1 ADTN, ADT2 ADTN )
```

ANSI Only--SQL3

```
  RETURNS BOOLEAN;
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
  RETURNS CHARACTER(1);
```

```
  RETURN
      ( ADT1 . C1 = ADT2 . C1
    AND ADT1 . C2 = ADT2 . C2
        .
        .
        .
    AND ADT1 . Cn = ADT2 . Cn );
```

b) If <equals function specification> specifies REF, then:

i) The <oid options> of the <abstract data type body> shall contain WITH OID.

ii) Let *ADTN* be the <abstract data type name>.

iii) The <abstract data type definition> contains an implicit <routine> with <routine name> *ADTN*_EQ that returns *true* if the object identified by the first operand and the object identified by the second operand are the same object (that is, they have the same object identifier):

24) If a <routine name> is specified as an <equals function specification> or <less-than function specification>, then Let *FD* be the <routine> identified by the <routine name>. The function referenced by *FD* is called the *equals function* or *less-than function*, respectively.

a) *FD* shall specify exactly two <operand data type>s, each of which shall specify *TN*.

b) The result data type of *FD* shall be BOOLEAN.

25) For every <cast clause> *CC*:

a) Let *ADTN* be the <abstract data type name> of the containing <abstract data type definition>. Let *ADT* be the abstract data type.

b) Let *ODT* and *RDT* be the <operand data type> and result data type of *CC*.

c) Exactly one of *ODT* and *RDT* shall be the same as *ADT*.

ANSI Only–SQL3

Let the other be *ADT2*.

d) *ADT2* shall not contain a <cast clause> specifying the same *ODT* and *RDT*.

ISO Only—caused by ANSI changes not yet considered by ISO

e) If both *ODT* and *RDT* are abstract data types, then *ODT* shall be the same as *ADT*.

Note: Thus a <cast clause> specifying CAST(*ADT1* AS *ADT2*) must be specified in *ADT1*. The requirement that this <cast clause> be specified in *ADT1* (rather than *ADT2*) is arbitrary. The essential requirement is that there should be at most one such <cast clause>, so that the CAST rules will be unambiguous.

---

f) Let *FD* be the <routine> identified by the <routine name> of *CC*. The SQL procedure referenced by *FD* is called the *cast function* of the <cast clause>.

g) *FD* shall specify exactly one <operand data type>, which shall be the same as *ODT*.

h) The result data type of *FD* shall be the same as *RDT*.

26) Case:

a) If the <abstract data type definition> contains a <routine> *ADTC* whose <routine name> is the <abstract data type name>, then the <routine> shall specify CONSTRUCTOR and the result data type of the <routine> shall be the <abstract data type name> of the containing <abstract data type definition>; this <routine> is the *default constructor function* for this ADT.

b) Otherwise, let *ADTN* be the <abstract data type name>; the <abstract data type definition> contains an implicit <routine> *ADTC*:

ANSI Only–SQL3

```
CONSTRUCTOR FUNCTION ADTN ( :C1 T1, . . . , :Cn Tn )
    RETURNS ADTN;
  DECLARE :TEMP ADTN;
  BEGIN
    NEW :TEMP;
    SET :TEMP.C1 := :C1;
    .
    .
    .
    SET :TEMP.Cn := :Cn;
    RETURN :TEMP
  END;
```

ISO Only—caused by ANSI changes not yet considered by ISO

```
CONSTRUCTOR FUNCTION ADTN ( :C1 T1, . . . , :Cn Tn )
RETURNS ADTN
:TEMP ADTN;
:TEMP.C1 := :C1;
    .
    .
    .
```

:TEMP.C*n* := :C*n*;
RETURN :TEMP
END FUNCTION

---

where, for each CONSTANT or UPDATABLE public column C*i* of the ADT, there is a corresponding parameter C*i* with the same name and the same data type.

27) Case:

a) If the <abstract data type definition> contains a <routine> *ADTD* whose <routine name> is REMOVE_*ADTN*, then the <routine> shall specify ''DESTRUCTOR'' and shall have exactly one parameter and that parameter shall be *ADTN*; this <routine> is the *default destructor function* for this ADT.

b) Otherwise, let *ADTN* be the <abstract data type name>; the <abstract data type definition> contains as its default destructor function an implicit <routine> *ADTD*:

ANSI Only–SQL3

```
DESTRUCTOR FUNCTION REMOVE_ADTN (:TEMP ADTN)
    RETURNS ADTN;
  BEGIN
    DESTROY :TEMP;
    RETURN NULL;
  END;
```

ISO Only—caused by ANSI changes not yet considered by ISO

DESTRUCTOR FUNCTION REMOVE_*ADTN* (:TEMP *ADTN*)
RETURNS *ADTN*
DESTROY :TEMP;
RETURN NULL;
END FUNCTION

---

28) The scope of the <column name>s and <routine name>s contained in the <member list> of the <abstract data type definition> shall be the entire <abstract data type definition>, including any <routine> contained in the <abstract data type definition>.

29) The descriptor of the abstract data type *ADT* defined by the <abstract data type definition> *ADTD* includes the name <abstract data type name>, the <oid options> specified, the column descriptor specified by each <column definition> in *ADTD*, and the routine descriptor specified by each <routine> in *ADTD*. The *i*-th column descriptor in the descriptor of *ADT* is given by the *i*-th <column definition> of *ADTD*.

Access Rules

1) The applicable privileges on each <abstract data type name> specified in <subtable clause> shall include UNDER.

General Rules

1) The <equals clause> and <less-than clause> specify functions that define the ordering referenced in the definitions of <predicate>s and the <order by clause>, and the equality reference in the <group by clause>, <having clause>, <unique predicate>, and DISTINCT. The equals function and less-than function are implicitly invoked as <boolean value expression>s. The equals function determines whether the first parameter is equal to the second parameter. The less-than function determines whether the first parameter is less than the second parameter.

2) The ordered sequence of column descriptors for stored columns defines the representation of an abstract data type comprising one or more components. Each component has a name and a data type. The value of an instance of an abstract data type is the ordered sequence of its stored columns, excluding any column with <column name> OID.

ISO Only—caused by ANSI changes not yet considered by ISO

3) The method and the time of binding of any referenced external procedures to the schema containing the <abstract data type definition> is implementation-defined.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <abstract data type body>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.46 <distinct type definition>

Function

Define a distinct abstract data type derived from another abstract data type.

Format

```
<distinct type definition> ::=
       CREATE DISTINCT TYPE <distinct type name> AS <data type>

<distinct type name> ::= <abstract data type name>
```

Syntax Rules

1) If a < distinct type definition> is contained in a <module> and if the <distinct type name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <module>.

2) The <qualified identifier> of <distinct type name> shall be different from the <qualified identifier> of any other <distinct type name> in the same schema.

3) <data type> shall not be a <predefined type>.
   Note: This is a temporary limitation.

4) A <distinct type definition> is *directly based on* the data type designated by <abstract data type name>. A <distinct type definition> *DTD* is *based on* a data type *DT* if *DTD* is directly based on *DT* or if *DTD* is directly based on a data type that is based on *DT*. No <distinct type definition> shall be based on itself.

Access Rules

1) If a <distinct type definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <distinct type name>.

General Rules

1) A <distinct type definition> defines an abstract data type based on the <abstract data type definition> identified by <abstract data type name>.

2) Let *DTN* be the <distinct type name> and *ADTN* be the <abstract data type name>.

3) Let *ADTD* be the <abstract data type definition> of the <abstract data type definition> identified by *ADTN* where each occurrence of ADTN is replaced by *DTN*.

4) The Syntax and General Rules for *ADTN*, except for rules creating the abstract data type descriptor and privilege descriptors, are applied without further Access Rule checking.

5) An abstract data type descriptor is created that describes the abstract data type being defined. The name contained in the abstract data type descriptor is *DTN*. The other values of the abstract data type descriptor are those produced by the General Rules of Subclause 11.45, "<abstract data type body>".

6) A privilege descriptor is created that defines the USAGE privilege for *DTN* to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the *DTN*. This privilege is grantable. The grantor for this privilege descriptor is set to the special grantor value ''_SYSTEM''.

7) If the applicable <privileges> on the abstract data type identified by *ADTN* include USAGE, then a privilege descriptor is created that defines the USAGE privilege for *DTN* to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the *DTN*. This privilege is grantable if the applicable <privilege> on the abstract data type identified by *ADTN* is grantable. The grantor for this privilege descriptor is set to the special grantor value ''_SYSTEM''.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <distinct type specification>.

2) The following restrictions apply for Intermediate SQL:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.47 <type template definition>

Function

Define a type template.

Format

```
<type template definition> ::=
       CREATE TYPE TEMPLATE <type template name>
         <template parameter declaration list>
         <abstract data type body>

<template parameter declaration list> ::=
       <left paren>
         <template parameter declaration>
           [ { <comma> <template parameter declaration>}... ]
       <right paren>

<template parameter declaration> ::=
       <template parameter name> <template parameter type>

<template parameter type> ::=
         <data type>
       | TYPE
```

Syntax Rules

1) If a <type template definition> is contained in a <schema definition>, and if the <type template name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

2) Each <type template definition> for which the <qualified identifier> of <type template name> is the same is said to belong to the same type template family.

3) The sequence of <template parameter type>s in the <template parameter declaration list> shall be different from that of any other <type template definition> in the same type template family.

4) The <template parameter name> of each <template parameter declaration> shall be different from the <template parameter name> of any other <template parameter declaration> in the containing <template parameter declaration list>.

5) The <data type> of a <template parameter type> shall not contain a <data type> that is a <template parameter name>.

6) No <template parameter name> in a <template parameter declaration> shall specify :GEN_ TYPE. :GEN_TYPE is a reserved <template parameter name> whose <template parameter type> is TYPE.

7) A <type template definition> *TTD* is directly based on a <type template definition> *TTD2* if *TTD* contains a <generated type reference> that references *TTD2*. A <type template definition> *TTD* is based on a <type template definition> *TTD2* if *TTD* is directly based on *TTD2* or if *TTD* is directly based on a <type template definition> that is based on *TTD2*. No <type template definition> shall be based on itself.

Access Rules

1) If a <type template definition> is contained in a <module>, then the <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <type template definition>.

General Rules

1) A <type template definition> defines a type template.

2) A type template descriptor is created that describes the type template *T* defined by the <type template definition>. The type template descriptor includes the <type template name>, <template parameter declaration list>, and <abstract data type body>.

3) Let *A* be the <authorization identifier> that owns *T*.

4) A privilege descriptor is created that specifies the privilege USAGE on this type template to *A*. This privilege is grantable if and only if all the applicable privileges required to define *T* are grantable. The grantor of this privilege descriptor is set to the special grantor value ''_SYSTEM''.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <type template definition>.

2) The following restrictions apply for Intermediate SQL:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.48 <drop type template statement>

Function

Drop a type template and all types generated by the type template.

Format

```
<drop type template statement> ::=
       DROP TYPE TEMPLATE <type template name> <drop behavior>
```

Syntax Rules

1) The <type template name> shall identify a <type template definition>.

2) Let *TT* be the type template identified by <type template name>.

3) If RESTRICT is specified, then no <abstract data type definition> generated from *TT* shall be referenced in any <table definition>, <view definition>, <abstract data type definition>, <module definition>,
ISO <external function definition>,
<routine>, or <type template definition>.

Access Rules

1) The <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <type template name>.

General Rules

1) Let *A* be the current <authorization identifer>. Let *TTN* be the <type template name>. The following <revoke statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

   REVOKE ALL PRIVILEGES ON *TTN* FROM *A* CASCADE

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop type template statement>.

2) The following restrictions apply for Intermediate SQL:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.49 <drop data type statement>

Function

Destroy an abstract data type.

Format

```
<drop data type statement> ::=
       DROP DATA TYPE <abstract data type name> <drop behavior>
```

Syntax Rules

1) The <abstract data type name> shall identify a <abstract data type definition>.

2) Let *D* be the abstract data type identified by the <abstract data type name> and let *DN* be that <abstract data type name>.

3) If RESTRICT is specified, then *D* shall not be referenced in any <table definition>, <view definition>, or <abstract data type definition>.

   Note: If CASCADE is specified, then such referenced objects will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

The <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <abstract data type name> identified by *DN*.

General Rules

1) Let *V* be the <table name> of any <view definition> that references a <column definition> that references *D*. The following <drop view statement> is effectively executed for each such <view definition> without further Access Rule checking:

   DROP VIEW *V* CASCADE

2) Let *T* be the <table name> of any <table definition> that references a <column definition> that references *D*. The following <drop table statement> is effectively executed for each such <table definition> without further Access Rule checking:

   DROP TABLE *T* CASCADE

3) Let *P* be the <external function name> of any <external function declaration> that references *D*. The following <drop external function statement> is effectively executed for each such <external function declaration> without further Access Rule checking:

   DROP EXTERNAL *P* CASCADE

4) Let *ADT* be the <abstract data type name> of any abstract data type that references *D*. The following <drop data type statement> is effectively executed for each such abstract data type without further Access Rule checking:

   DROP DATA TYPE *ADT* CASCADE

5) Let *M* be the <module name> of any <module> that references *D*. The following <drop module statement> is effectively executed for each such <module> without further Access Rule checking:

   DROP MODULE *M* CASCADE

6) For every privilege descriptor that references *D*, the following <revoke statement> is effectively executed:

   REVOKE *PRIV* ON *D* FROM *GRANTEE* CASCADE

   where *PRIV* and *GRANTEE* are respectively the action and grantee in the privilege descriptor.

7) The identified abstract data type is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop data type statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.50 <module definition>

Function

Define a module.

Format

```
<module definition> ::=
       CREATE MODULE <module name>
         <module remainder>
       END MODULE
```

Syntax Rules

1) Case:

   a) If a <module definition> is contained in a <schema definition>, and if the <module definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

   b) If a <module definition> is contained in a <module> other than in a <schema definition>, and if the <module definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <module>.

2) The <qualified identifier> of <module name> shall be different from the <qualified identifier> of any other <module name> of a module definition> in the same schema.

Access Rules

1) If a <module definition> is contained in a <module>, then the <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <module name>.

General Rules

1) A <module definition> defines a module.

2) A privilege descriptor is created that defines the EXECUTE privilege on this module to the <authorization identifier> of the <schema> or <module> in which the <module definition> appears. This privilege is grantable. The grantor of the privilege descriptor is set to the special grantor value ''_SYSTEM''.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <module definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.51 <drop module statement>

Function

Destroy a module.

Format

```
<drop module statement> ::=
       DROP MODULE <module name> <drop behavior>
```

Syntax Rules

1) Let *M* be the module identified by <module name> and let *MN* be the <module name>.

Access Rules

1) The <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the module identified by *MN*.

General Rules

1) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

   REVOKE EXECUTE ON MODULE *M* FROM *A*

2) The descriptor of *M* is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop module statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.52 <grant statement>

Function

Define privileges.

Format

```
<grant statement> ::=
       GRANT <privileges>
         TO <grantee> [ { <comma> <grantee> }... ]
           [ WITH GRANT OPTION ]
```

Syntax Rules

1) Let *O* be the object identified by the <object name> contained in <privileges>.

2) Let *A* be the current <authorization identifier>. For each <grantee> specified in <privileges>, a set of privilege descriptors is identified. The privilege descriptors identified are those defining, for each <action> explicitly or implicitly in <privileges>, that <action> on *O* held by *A* with grant option.

3) If one or more <grantee>s is a role, then the WITH GRANT OPTION and <privileges> containing the <action> REFERENCES shall not be specified.

4) Grants of ALL PRIVILEGES and ALL SCHEMA PRIVILEGES <privileges> to role <grantee>s do not include the <action>s REFERENCES or USAGE.

Access Rules

1) The applicable privileges shall include a privilege identifying *O*.

General Rules

1) The <privileges> specify one or more privileges on the schema object as follows:

   a) ALL SCHEMA PRIVILEGES specifies all privileges on all <schema element>s in the containing schema, including <schema element>s subsequently added by <SQL schema statement>.

   b) The <object privileges> specify one or more privileges on the object identified by the <object name>.

   c) EXTERNAL PRIVILEGES specifies the external call privilege on each external function identified by an <external function name>.

2) For every identified privilege descriptor, a privilege descriptor is created that specifies the identical <grantee>, <action>, object *O*, and grantor *A*. Let *CPD* be the set of privilege descriptors created.

3) For every identified privilege descriptor whose action is SELECT, INSERT, UPDATE, or REFERENCES without a column name, privilege descriptors are also created for each column *C* in *O* for which *A* holds the corresponding privilege with grant option. For each such

column, a privilege descriptor is created that specifies the identical <grantee>, the identical <action>, object *C*, and grantor *A*.

If <privileges> contains an <action> that is DELETE or TRIGGER, then a table privilege descriptor is created. If <privileges> contains an <action> that is SELECT, INSERT, UPDATE, or REFERENCES without a <privilege column list>, then a table privilege descriptor is created along with a column privilege descriptor for each column in the table identified by <table name>. If <privileges> contains an <action> that is SELECT, INSERT, UPDATE, or REFERENCES with a <privilege column list>, then a column privilege descriptor is created for each column in the <privilege column list>. If <privileges> contains an <action> that is USAGE, then a usage privilege descriptor is created.

4) If WITH GRANT OPTION was specified, each privilege descriptor also indicates that the privilege is grantable.

5) If <table name> is specified, then let *T* be the table identified by the <table name>.

6) For every updatable view *V* owned by some grantee *G* such that *T* is some leaf underlying table of the <query expression> of *V*:

   a) Let *VN* be the <table name> of *V*.

   b) If WITH GRANT OPTION is specified, then let *WGO* be ''WITH GRANT OPTION''; otherwise, let *WGO* be a zero-length string.

   c) For every privilege descriptor *PD* in *CPD*, let *PA* be the action included in *PD*.

      i) If *PA* is INSERT, UPDATE, or DELETE, then the following <grant statement> is effectively executed as though the current <authorization identifier> were ''_SYSTEM'' and without further Access Rule checking:

         GRANT *PA* ON *VN* TO *G WGO*

      ii) If *PA* is *A*(*CT*), where *A* is INSERT, UPDATE, or SELECT and *CT* is the name of some column of *T* such that there is a corresponding column in *V*, named *CVN*, that is derived from *CT*, then the following <grant statement> is effectively executed as though the current <authorization identifier> were ''_SYSTEM'' and without further Access Rule checking:

         GRANT *A*(*CVN*) ON *VN* TO *G WGO*

7) For every <grantee> *G* and for every view *V1* owned by *G*, if *G* has been granted SELECT privilege WITH GRANT OPTION on all tables identified by a <table name> contained in the <query expression> of *V1*, then for every privilege descriptor with a <privileges> *P* that contains SELECT, a <grantor> of ''_SYSTEM'', <object> of *V1*, and <grantee> *G* that is not grantable, the following <grant statement> is effectively executed <grant statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

   GRANT *P* ON *V1* TO *G* WITH GRANT OPTION

8) For every <grantee> *G* and for every domain *D1* owned by *G*, if *G* has been granted REFERENCES privilege WITH GRANT OPTION on every column referenced in the <search condition> included in a domain constraint descriptor included in the domain descriptor of *D1* and a grantable USAGE privilege on all domains, character sets, collations, and translations whose <domain name>s, <character set name>s, <collation name>s, and <translation name>s, respectively, are included in the domain descriptor, then for every privilege descriptor with

<privileges> USAGE, a <grantor> of ''_SYSTEM'', <object> *D1*, and <grantee> *G* that is not grantable, the following <grant statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

GRANT USAGE ON DOMAIN *D1* TO *G* WITH GRANT OPTION

9) For every <grantee> *G* and for every collation *C1* owned by *G*, if the USAGE privilege of *G* for the character set name included in the collation descriptor of *C1* is grantable and if the USAGE privilege of *G* for the translation name, if any, included in the collation descriptor of *C1* is grantable, then for every privilege descriptor with a <privileges> *P*, a <grantor> of ''_SYSTEM'', <object> of *C1*, and <grantee> *G* that is not grantable, the following <grant statement> is effectively executed with a current <authorization identifier> of ''_SYSTEM'' and without further Access Rule checking:

GRANT USAGE ON COLLATION *C1* TO *G* WITH GRANT OPTION

10) For every <grantee> *G* and for every translation *T1* owned by *G*, if the USAGE privilege of *G* for every character set identified by a <character set specification> contained in the <translation definition> of *T1* is grantable, then for every privilege descriptor with a <privileges> *P*, a <grantor> of ''_SYSTEM'', <object> of *T1*, and <grantee> *G* that is not grantable, the following <grant statement> is effectively executed as though the current <authorization identifier> were ''_SYSTEM'' and without further Access Rule checking:

GRANT *P* ON TRANSLATION *T1* TO *G* WITH GRANT OPTION

11) If <table name> is specified, then for each view *V* owned by some <grantee> *G* such that *T* or some column *CT* of *T* is referenced in the <query expression> of *V*, let RT_i, for *i* ranging from 1 to the number of tables identified by the <table reference>s contained in the <query expression> of *V*, be the <table name>s of those tables. For every column *CV* of *V*:

a) Let CRT_{ij}, for *j* ranging from 1 to the number of columns of RT_i that are underlying columns of *CV*, be the <column name>s of those columns.

b) If WITH GRANT OPTION was specified, then let *WGO* be ''WITH GRANT OPTION''; otherwise, let *WGO* be a zero-length string.

c) If, following successful execution of the <grant statement>, *G* will have REFERENCES(CRT_{ij}) for all *i* and for all *j*, and will have REFERENCES on some column of RT_i for all *i*, then the following <grant statement> is effectively executed as though the current <authorization identifier> were ''_SYSTEM'' and without further Access Rule checking:

GRANT REFERENCES *CV* ON *V* TO *G WGO*

12) If two privilege descriptors are identical except that one indicates that the privilege is grantable and the other indicates that the privilege is not grantable, then both privilege descriptors are set to indicate that the privilege is grantable.

13) Redundant duplicate privilege descriptors are removed from the multiset of all privilege descriptors.

14) For every combination of <grantee> and <action> on *O* specified in <privileges>, if there is no corresponding privilege descriptor in the set of identified privilege descriptors, then a completion condition is raised: *warning—privilege not granted*.

15) If ALL PRIVILEGES was specified, then for each grantee, if no privilege descriptors were identified, then a completion condition is raised: *warning—privilege not granted*.

16) The privileges defined for *T* do not affect the privileges of any supertable or any subtable of *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

ANSI Only–SQL3

a) In Conforming Full SQL language, an <object name> shall not specify SET, or LIST.

ISO Only—caused by ANSI changes not yet considered by ISO

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) In Conforming Intermediate SQL language, an <object name> shall not specify COLLATION or TRANSLATION.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) In Conforming Entry SQL language, an <object name> shall not specify TABLE.

   b) In Conforming Entry SQL language, an <object name> shall not specify CHARACTER SET or DOMAIN.

11.53 <role definition>

Function

Define a role.

Format

```
<role definition> ::= CREATE ROLE <role name>
```

Syntax Rules

1) The specified <role name> shall be different from any other <role name> or <authorization identifier> in the SQL environment.

Access Rules

1) The Access Rules are implementation-defined.

General Rules

1) A <role definition> defines a role.

2) Let *B* be the <authorization identifier> of the current SQL-session. A role authorization descriptor is created that identifies that the role identified by <role name> has been granted to *B* with the WITH ADMIN OPTION.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <role definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.54 <grant role statement>

Function

Define role authorizations.

Format

```
<grant role statement> ::=
       GRANT <role granted> [ { <comma> <role granted> }... ]
         TO <grantee> [ { <comma> <grantee> }... ]
           [ WITH ADMIN OPTION ]

<role granted> ::= <role name>
```

Syntax Rules

1) No role identified by a specified <grantee> shall be contained in any role identified by a specified <role granted>; that is, no cycles of role grants are allowed.

Access Rules

1) The applicable <authorization identifier> shall have all roles identified by the specified <role granted>s as applicable roles with the WITH ADMIN OPTION.

General Rules

1) For every <grantee> specified, a set of role authorization descriptors is created that defines the grant of each role identified by a <role granted> to the <grantee>.

2) If WITH ADMIN OPTION is specified, then each role authorization descriptor also indicates that the role is grantable with the WITH ADMIN OPTION.

3) If two role authorization descriptors are identical except that one indicates that the role is grantable with the WITH ADMIN OPTION and the other indicates that the role is not, then both role authorization descriptors are set to indicate that the role is grantable with the WITH ADMIN OPTION.

4) Redundant duplicate role authorization descriptors are removed from the multiset of all role authorization descriptors.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <grant role statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.55 <revoke role statement>

Function

Destroy role authorizations.

Format

```
<revoke role statement> ::=
       REVOKE <role revoked> [ { <comma> <role revoked> }... ]
         FROM <grantee> [ { <comma> <grantee> }... ]

<role revoked> ::= <role name>
```

Syntax Rules

1) For every <grantee> specified, the set of role authorization descriptors is identified that defines the grants of the specified <role revoked>s to the <grantee>.

Access Rules

1) The <authorization identifier> of the current SQL-session shall have the WITH ADMIN OPTION on each <role name> specified as a <role revoked>.

General Rules

1) The identified role authorization descriptors are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <revoke role statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.56 <drop role statement>

Function

Destroy a role.

Format

```
<drop role statement> ::= DROP ROLE <role name>
```

Syntax Rules

None.

Access Rules

1) The <authorization identifier> of the current SQL-session shall have been granted role *R* with the WITH ADMIN OPTION.

General Rules

1) Let *R* be the role identified by the specified <role name>.

2) Let *A* be any <authorization identifier> or role identified by a role authorization descriptor as having been granted *R*.

3) The following <revoke role statement> is effectively executed:

   REVOKE *R* FROM *A*

4) The identified role and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop role statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.57 <revoke statement>

Function

Destroy privileges.

Format

```
<revoke statement> ::=
       REVOKE [ GRANT OPTION FOR ] <privileges>
         FROM <grantee> [ { <comma> <grantee> }... ] <drop behavior>
```

Syntax Rules

1) If <object name> specifies a <domain name>, <type template name>, <character set name>, <collation name>, or <translation name>, then <privileges> shall specify USAGE. If <object name> specifies an <abstract data type name>, then <privileges> may specify USAGE. Otherwise, USAGE shall not be specified.

2) SELECT is equivalent to specifying both the SELECT table privilege and SELECT (<privilege column list>) for all columns of <table name>.

3) INSERT is equivalent to specifying both the INSERT table privilege and INSERT (<privilege column list>) for all columns of <table name>.

4) UPDATE is equivalent to specifying both the UPDATE table privilege and UPDATE (<privilege column list>) for all columns of <table name>.

5) REFERENCES is equivalent to specifying both the REFERENCES table privilege and REFERENCES (<privilege column list>) for all columns of <table name>.

6) Let *O* be the object identified by the <object name>.

7) Let *A* be the current <authorization identifier>. For every <grantee> specified, a set of privilege descriptors is identified. A privilege descriptor *P* is said to be *identified* if it belongs to the set of privilege descriptors that define, for any <action> explicitly or implicitly in <privileges>, that <action> on *O* granted by *A* to <grantee>.

   Note: Column privilege descriptors become identified when <action> explicitly or implicitly contains a <privilege column list>.

8) A privilege descriptor *D* is *allowed to be created* by a grant permitted by *P* if either:

   a) The following conditions hold:

      i) *P* indicates that the privilege that it represents is grantable, and

      ii) The grantee of *P* is the same as the grantor of *D* or the grantee of *P* is PUBLIC, and

      iii) Case:

         1) *P* and *D* are both column privilege descriptors. The action and the identified column of *P* are the same as the action and identified column of *D*, respectively.

2) *P* is a table privilege descriptor and *D* is a column privilege descriptor. The identified table of (P) is the same as the identified table of *D* and the action of *P* is the same as the action of *D* and the action of *P* is SELECT. LE>Neither *P* nor *D* are column privilege descriptors. The action and the identified table, domain, character set, collation, or translation of *P* are the same as the action and the identified table, domain, character set, collation, or translation of *D*, respectively.

b) The following conditions hold:

i) The privilege descriptor for *D* indicates that its grantor is the special grantor value ''_SYSTEM'', and

ii) The action of *P* is the same as the action of *D*, and

iii) The grantee of *P* is the owner of the table, collation, or translation identified by *D*, or the grantee of *P* is PUBLIC, and

iv) One of the following conditions hold:

1) *P* and *D* are both table privilege descriptors, the privilege descriptor for *D* identifies the <table name> of a <view definition> *V* and either:

A) The action of *P* is SELECT and the identified table of *P* is contained in the <query expression> of *V*, or

B) *V* is an updatable view and the identified table of *P* is the underlying table of the <query expression>.

2) *P* and *D* are both column privilege descriptors, the privilege descriptor *D* identifies a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V* and either:

A) *V* is an updatable view. For every column *CV* identified by a <column name> *CVN*, there is a corresponding column in the underlying table of the <query expression> *TN*. Let *CTN* be the <column name> of the column of the <query expression> from which *CV* is derived. The action for *P* is UPDATE or INSERT and the identified column of *P* is *TN.CTN*; or

B) For every table *T* identified by a <table reference> contained in the <query expression> of *V* and for every column *CT* that is a column of *T* and an underlying column of *CV*, the action for *P* is REFERENCES and either the identified column of *P* is *CT* or the identified table of *P* is *T*.

3) *P* is a table privilege descriptor and the column privilege descriptor *D* identifies a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V*. Let *TN* be a <table name> contained in the <query expression> of the view. The action for *P* is SELECT and the identified table of *P* is *TN*.

4) The privilege descriptor *D* identifies the <collation name> of a <collation definition> *CO* and the identified character set name of *P* is included in the collation descriptor for *CO* or the identified translation name of *P* is included in the collation descriptor for *CO*.

5) The privilege descriptor *D* identifies the <translation name> of a <translation definition> *TD* and the identified character set name of *P* is contained in the <source character set specification> or the <target character set specification> immediately contained in *TD*.

c) The following conditions hold:

i) The privilege descriptor for *D* indicates that its grantor is the special grantor value ''_SYSTEM'';

ii) The grantee of *P* is the owner of the domain identified by *D*, or the grantee of *P* is PUBLIC, and

iii) The privilege descriptor *D* identifies the <domain name> of a <domain definition> *DO* and either the column privilege descriptor *P* has an action of REFERENCES and identifies a column referenced in the <search condition> included in the domain descriptor for *DO*, or the privilege descriptor *P* has an action of USAGE and identifies a domain, collation, character set, or translation whose <domain name>, <collation name>, <character set name> or <translation name> respectively is contained in the <search condition> of the domain descriptor for *DO*.

9) A privilege descriptor *D* is said to be *directly dependent on* another privilege descriptor *P* if *D* represents a privilege allowed to be created by a grant permitted by *P*.

10) The *privilege dependency graph* is a directed graph such that:

a) Each node represents a privilege descriptor, and

b) Each arc from node *P1* to node *P2* represents the fact that *P2* directly depends on *P1*.

An *independent node* is one that has no incoming arcs.

11) A privilege descriptor *P* is said to be *modified* if either *P* is a SELECT column privilege descriptor and a SELECT table privilege descriptor with the same grantee, grantor, catalog name, schema name, and table name is a modified privilege descriptor, or:

a) *P* indicates that the privilege that it represents is grantable, and

b) *P* directly depends on an identified privilege descriptor or a modified privilege descriptor, and

c) Case:

i) The following conditions hold:

1) *P* is not a REFERENCES column privilege descriptor that identifies a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V*, and;

2) Let *XO* and *XA* respectively be the identifier of the object identified by a privilege descriptor *X* and the action of *X*. Within the set of privilege descriptors upon which *P* directly depends, there exist some *XO* and *XA* for which the set of identified privilege descriptors unioned with the set of modified privilege descriptors include all privilege descriptors specifying the grant of *XA* on *XO* WITH GRANT OPTION.

ii) The following conditions hold:

1) *P* is a REFERENCES column privilege descriptor that identifies a column *CV* named by a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V*; and

2) Let *SP* be the set of privileges upon which *P* directly depends. For every table *T* identified by a <table reference> contained in the <query expression> of *V*, let *RT* be the <table name> of *T*. There exists a column *CT* whose <column name> is *CRT*, such that:

A) *CT* is a column of *T* and an underlying column of *CV*, and

B) Every privilege descriptor *PD* that is the descriptor of some member of *SP* that specifies REFERENCES on *CRT* WITH GRANT OPTION is either an identified privilege descriptor for *CRT* or a modified privilege descriptor for *CRT*.

3) GRANT OPTION FOR is not specified and within the set of privileges upon with *P* directly depends, either:

A) There exists some RT_i for which the set of identified privilege descriptors unioned with the set of abandoned privilege descriptors include all column privilege descriptors specifying the grant of REFERENCES on RT_i; or

B) There exists some CRT_{ij} for which the set of identified privilege descriptors unioned with the set of abandoned privilege descriptors include all column privilege descriptors specifying the grant of REFERENCES on CRT_{ij}.

and

d) At least one of the following is true:

i) GRANT OPTION FOR is specified and the grantor of *P* is the special grantor value ''_SYSTEM''.

ii) There exists a path to *P* from an independent node that includes no identified or modified privilege descriptors. *P* is said to be a *marked modified privilege descriptor*.

iii) *P* directly depends on a marked modified privilege descriptor, and the grantor of *P* is the special grantor value ''_SYSTEM''. *P* is said to be a *marked modified privilege descriptor*.

12) A privilege descriptor *P* is *abandoned* if:

a) It is not an independent node, and

Case:

i) GRANT OPTION FOR is not specified, *P* is not itself a modified privilege descriptor, and there exists no path to *P* from any independent node other than paths that include an identified privilege descriptor or a modified privilege descriptor.

ii) GRANT OPTION FOR is specified, *P* is not itself a modified privilege descriptor, and there exists no path to *P* from any independent node other than paths that include a modified privilege descriptor.

b) *P* is a SELECT column privilege descriptor and there exists a SELECT table privilege descriptor *X* with the same grantee, grantor, catalog name, schema name, and table name and

Case:

i) GRANT OPTION FOR is not specified and *X* is an identified privilege descriptor or an abandoned privilege descriptor.

ii) GRANT OPTION FOR is specified and *X* is an abandoned privilege descriptor.

c) The following conditions hold:

i) *P* is a REFERENCES column privilege descriptor that identifies a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V*; and

ii) GRANT OPTION FOR is not specified, and, letting *SP* be the set of privileges upon which *P* directly depends, either:

1) There exists some table name *RT* such that:

A) *RT* is the name of the table identified by some <table reference> contained in the <query expression> of *V*, and

B) For every column privilege descriptor *CPD* that is the descriptor of some member of *SP* that specifies REFERENCES on *RT*, *CPD* is either an identified privilege descriptor for *RT* or an abandoned privilege descriptor for *RT*.

or

2) There exists some column name *CRT* such that:

A) *CRT* is the name of some column of the table of some <table reference> contained in the <query expression> of *V*, and

B) For every column privilege descriptor *CPD* that is the descriptor of some member of *SP* that specifies REFERENCES on *CRT*, *CPD* is either an identified privilege descriptor for *CRT* or an abandoned privilege descriptor for *CRT*.

13) Let *S1* be the name of any schema and let *A1* be the applicable <authorization identifier> for privilege determination with the schema identified by *S1*.

14) Let *V* be any view descriptor included in *S1*. *V* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having SELECT privilege on one or more tables or USAGE privilege on one or more domains, abstract data types, collations, character sets, or translations or EXTERNAL PRIVILEGES on one or more of the external function invocations whose names are contained in the <query expression> of *V*.

15) Let *TC* be any table constraint descriptor included in *S1*. *TC* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having REFERENCES privilege on one or more referenced columns of *TC* or USAGE privilege on one or more domains, abstract data types, collations, character sets, or translations or EXTERNAL PRIVILEGES on one or more of the external function invocations whose names are contained in any <search condition> of *TC*.

****Editor's Note****
The fact that the preceing rule does not address the SELECT privilege has been identified as a Possible Problem. See Possible Problem **141** in the Editor's Notes.

16) Let *AX* be any assertion descriptor included in *S1*. *AX* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having REFERENCES privilege on one or more referenced columns of *AX* or USAGE privilege on one or more domains, abstract data types, collations, character sets, or translations or EXTERNAL PRIVILEGES on one or more of the external function invocations whose names are contained in any <search condition> of *AX*.

17) Let *TR* be any trigger descriptor included in *S1*. *TR* is said to be *abandoned* if the destruction of all identified privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having REFERENCES privilege on one or more referenced columns of *AX* or USAGE privilege on one or more of the domains, abstract data types, collations, character sets, or translations if any, whose names are contained in any <search condition> of *TR*, the appropriate privileges for any <triggered SQL statement> appearing in *TR*, SELECT privilege on the subject table of *TR*, SELECT privilege on one or more tables appearing in any <search condition> of *TR*, or EXTERNAL PRIVILEGES on one or more of the external function invocations whose names are contained in any <search condition> of *TR*.

18) Let *DC* be any domain constraint descriptor included in *S1*. *DC* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having REFERENCES privilege on one or more referenced columns of *DC* or USAGE privilege on one or more domains, abstract data types, collations, character sets, or translations whose names are contained in any <search condition> of *DC*.

19) For every domain descriptor *DO* contained in *S1*, *DO* is said to be *lost* if the destruction of all abandoned privileged descriptors, and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having EXTERNAL PRIVILEGES on one or more of the external function invocations appearing in any <search condition> of *DO*, or USAGE privilege on any character set included in the data type descriptor included in *DO*.

20) For every table descriptor *TD* contained in *S1*, for every column descriptor *CD* included in *TD*, *CD* is said to be *lost* if either:

    a) The destruction of all abandoned privileged descriptors, and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having USAGE privilege on any character set included in the data type descriptor included in *CD*, or

    b) The name of the domain *DN* included in *CD*, if any, identifies a lost domain descriptor and the destruction of all abandoned privileged descriptors, and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having USAGE privilege on any character set included in the data type descriptor of the domain descriptor of *DN*.

21) For every module *MO*, let *G* be the <module authorization identifier> that owns *MO*. *MO* is said to be *lost* if the destruction of all abandoned privileged descriptors, and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *G* no longer having USAGE privilege on the character set referenced in the <module character set specification> of *MO*.

22) Let *DT* be an abstract data type whose descriptor is in *S*. *DT* is said to be *abandoned* if the destruction of all identified privilege descriptors and abandoned privilege descriptors would result in <authorization identifier> *A* no longer having any USAGE or UNDER privilege on one or more columns of a table or any abstract data type.

23) For every <type template definition> *TT* contained in *S1*, *TT* is said to be abandoned if the destruction of all identified privilege descriptors and all abandoned privilege descriptors, would result in *A1* no longer having some privilege required to create the <type template definition>.

24) Let *SD* be the descriptor of the schema *S1*. *SD* is said to be *lost* if the destruction of all abandoned privileged descriptors, and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having USAGE privilege on the default character set included in the schema descriptor *SD*.

25) For every domain descriptor *DO* contained in *S1*, *DO* is said to be *impacted* if *DO* is not lost, and the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having USAGE privilege on the collation whose name is contained in the <collate clause> of *DO*.

26) For every column descriptor *CD* contained in a table descriptor contained in *S1*, *CD* is said to be *impacted* if *CD* is not lost, and the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having USAGE privilege on the collation whose name is contained in the <collate clause> of *CD*.

27) For every collation descriptor *CN* contained in *S1*, *CN* is said to be *impacted* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having USAGE privilege on the collation whose name is contained in the <collation source> of *CN*.

28) For every character set descriptor *CSD* contained in *S1*, *CSD* is said to be *impacted* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having USAGE privilege on the collation whose name is contained in *CSD*.

29) If RESTRICT is specified, then there shall be no abandoned privilege descriptors, abandoned views, abandoned table constraints, abandoned assertions, abandoned domain constraints, lost domains, lost columns, lost modules, lost schema, impacted domains, impacted columns, impacted collations, impacted character sets, or abandoned abstract data types.

30) If CASCADE is specified, then the impact on a module that is determined to be a lost module is implementation-defined.

Access Rules

1) The applicable privileges shall include a privilege identifying *O*.

General Rules

1) If GRANT OPTION FOR is not specified, then:

   a) All abandoned privilege descriptors are destroyed.

   b) The identified privilege descriptors are destroyed.

   c) The modified privilege descriptors are set to indicate that they are not grantable.

2) If GRANT OPTION FOR is specified, then

   Case:

   a) If CASCADE is specified, then all abandoned privilege descriptors are destroyed.

   b) Otherwise, if there are any privilege descriptors directly dependent on an identified privilege descriptor that are not modified privilege descriptors, then an exception condition is raised: *dependent privilege descriptors still exist*.

   The identified privilege descriptors and the modified privilege descriptors are set to indicate that they are not grantable.

3) For every abandoned view descriptor *V*, let *S1.VN* be the <table name> of *V*. The following <drop view statement> is effectively executed without further Access Rule checking:

   DROP VIEW *S1.VN* CASCADE

4) For every abandoned table constraint descriptor *TC*, let *S1.TCN* be the <constraint name> of *TC* and let *S2.T2* be the <table name> of the table that contains *TC* (*S1* and *S2* not necessarily different). The following <alter table statement> is effectively executed without further Access Rule checking:

   ALTER TABLE *S2.T2* DROP CONSTRAINT *S1.TCN* CASCADE

5) For every abandoned assertion descriptor *AX*, let *S1.AXN* be the <constraint name> of *AX*. The following <drop assertion statement> is effectively executed without further Access Rule checking:

   DROP ASSERTION *S1.AXN*

6) For every abandoned trigger descriptor *TR*, let *S1.TRN* be the <trigger name> of *TR*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

   DROP TRIGGER *S1.TRN*

7) For every abandoned domain constraint descriptor *DC*, let *S1.DCN* be the <constraint name> of *DC* and let *S2.DN* be the <domain name> of the domain that contains *DC*. The following <alter domain statement> is effectively executed without further Access Rule checking:

   ALTER DOMAIN *S2.DN* DROP CONSTRAINT *S1.DCN*

8) For every lost column descriptor *CD*, let *S1.TN* be the <table name> of the table whose descriptor includes the descriptor *CD* and let *CN* be the <column name> of *CD*. The following <alter table statement> is effectively executed without further Access Rule checking:

   ALTER TABLE *S1.TN* DROP COLUMN *CN* CASCADE

9) For every lost domain descriptor *DO*, let *S1.DN* be the <domain name> of *DO*. The following <drop domain statement> is effectively executed without further Access Rule checking:

   DROP DOMAIN *S1.DN* CASCADE

10) For every lost schema descriptor *SD*, the default character set of that schema is modified to include the name of the implementation-defined <character set specification> that would have been this schema's default character set had the <schema definition> not specified a <schema character set specification>.

11) If the object identified by *O* is a collation, let *OCN* be the name of that collation.

12) For every impacted domain descriptor *DO*, *DO* is modified such that it does not include *OCN*.

13) For every impacted column descriptor *CD*, *CD* is modified such that it does not include *OCN*.

14) For every impacted collation descriptor *CN*, *CN* is modified such that it does not include *OCN*. If *CN* does not include any translation name, then *CN* is modified to indicate that it utilizes the DEFAULT collation for its character repertoire.

15) For every impacted character set descriptor *CSD*, *CSD* is modified such that it does not include *OCN*. If *CSD* does not include any translation name, then *CSD* is modified to indicate that it utilizes the DEFAULT collation for its character repertoire.

16) For every abandoned abstract data type *DT* with <abstract data type name> *S1.DTN*, the following <drop data type statement> is effectively executed without further Access Rule checking:

DROP DATA TYPE *S1.DTN* CASCADE

17) For every abandoned type template *TT* with <type template name> *S1.TTN*, the following <drop type template statement> is effectively executed without further Access Rule checking:

DROP TYPE *S1.TTN* CASCADE

18) For every combination of <grantee> and <action> on *O* specified in <privileges>, if there is no corresponding privilege descriptor in the set of identified privilege descriptors, then a completion condition is raised: *warning—privilege not revoked*.

19) If ALL PRIVILEGES was specified, then for each <grantee>, if no privilege descriptors were identified, then a completion condition is raised: *warning—privilege not revoked*.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain a <revoke statement>.

12 Standard type templates and types

12.1 Introduction

Standard type templates and types are defined in this Clause by using the structure of <type template definition> and <abstract data type definition> to describe their interfaces. The semantics of their public <function declaration>s and <column definition>s will generally be provided informally.

These standard type templates and types are informally grouped into interrelated families, with the type template and type names of a family all sharing a common prefix ending with an underscore.

Within the Subclause defining such a family, any concepts and terminology specific to that family will be included, for example where the concepts and terminology correspond to those of some other standard.

ANSI Only–SQL3

12.2 SQL type templates and types

The type templates and types in this family are designed to correspond to the functionality of parts of SQL itself.

The SQL_Table template defines the functions SQL_Insert, SQL_Select, SQL_Update, and SQL_Delete to correspond to the data manipulation language statements INSERT, SELECT, UPDATE, and DELETE. Other functions correspond to predicates and operators on tables.

The SQL_Set template inherits the functionality of an SQL_Table, and imposes the constraint that a set cannot contain duplicate elements. It also defines a Subset function.

The SQL_List template also inherits the functionality of an SQL_Table, refines the semantics of some inherited functions, and introduces additional functions Element and Position related to the ordering of elements in the list.

The SQL_Empty_Table, SQL_Empty_Set, and SQL_Empty_list types define tables, sets, and lists that are always empty, and are thus not constrained by any element type.

12.2.1 Definitions used in this Subclause

To Be Supplied

12.2.2 Concepts used in this Subclause

To Be Supplied

ANSI Only–SQL3

12.2.3 SQL_Table type template

Function

SQL_Table provides functions to carry out insertion, selection, updating, and deletion on SQL tables.

Definition

```
CREATE TYPE TEMPLATE  SQL_Table ( :Element_Type TYPE )

          !! default constructor and destructor signatures are to be
          !! inserted here when the naming problem is resolved

      (

      CAST ( :GEN_TYPE AS :Element_Type WITH Table_to_Element ) ; ,

      CAST ( :GEN_TYPE AS SQL_Set ( :Element_Type ) WITH Distinct ) ; ,

      CAST ( :GEN_TYPE AS SQL_List ( :Element_Type ) WITH Table_to_List ) ; ,

      CAST ( SQL_Empty_Table AS :GENTYPE WITH Empty_Table_to_Table ) ; ,

      FUNCTION SQL_Insert ( :tabref REF ( :GEN_TYPE ) ,
                            :stmt CHARACTER VARYING (max_stmt_length) ,
                            :descr_name CHARACTER VARYING (max_name_length) ) ; ,

      FUNCTION SQL_Select ( :tabref REF ( :GEN_TYPE ) ,
                            :stmt CHARACTER VARYING (max_stmt_length) ,
                            :descr_name CHARACTER VARYING (max_name_length) )
               RETURNS SQL_Table(Object) ; ,  !! assuming Object is a supertype of all
      ADT

      FUNCTION SQL_Update ( :tabref REF ( :GEN_TYPE ) ,
                            :stmt CHARACTER VARYING (max_stmt_length) ,
                            :descr_name CHARACTER VARYING (max_name_length) ) ; ,

      FUNCTION SQL_Delete ( :tabref REF ( :GEN_TYPE ) ,
                            :stmt CHARACTER VARYING (max_stmt_length) ,
                            :descr_name CHARACTER VARYING (max_name_length) ) ; ,

      FUNCTION In ( :table :GEN_TYPE, :elem :Element_Type )
               RETURNS BOOLEAN ; ,

      FUNCTION Exists ( :table :GEN_TYPE ) RETURNS BOOLEAN ; ,

      FUNCTION Unique ( :table :GEN_TYPE ) RETURNS BOOLEAN ; ,

      FUNCTION For_Some ( :table :GEN_TYPE,
                          :pred CHARACTER VARYING (max_pred_length) ,
                          :descr_name CHARACTER VARYING (max_name_length) )
               RETURNS BOOLEAN ; ,
```

```
FUNCTION For_All ( :table :GEN_TYPE ,
                   :pred CHARACTER VARYING (max_pred_length) ,
                   :descr_name CHARACTER VARYING (max_name_length) )
         RETURNS BOOLEAN ; ,

FUNCTION Average ( :table :GEN_TYPE ) RETURNS :Element_Type ; ,

FUNCTION Maximum ( :table :GEN_TYPE ) RETURNS :Element_Type ; ,

FUNCTION Minimum ( :table :GEN_TYPE ) RETURNS :Element_Type ; ,

FUNCTION Sum ( :table :GEN_TYPE ) RETURNS :Element_Type ; ,

FUNCTION Count ( :table :GEN_TYPE ) RETURNS count_type ; ,

FUNCTION Distinct ( :table :GEN_TYPE ) RETURNS SQL_Set ( :Element_Type ) ; ,

FUNCTION Union ( :table :GEN_TYPE, :table2 :GEN_TYPE )
         RETURNS :GEN_TYPE ; ,

FUNCTION Union_All ( :table :GEN_TYPE, :table2 :GEN_TYPE )
         RETURNS :GEN_TYPE ; ,

FUNCTION Except ( :table :GEN_TYPE, :table2 :GEN_TYPE )
         RETURNS :GEN_TYPE ; ,

FUNCTION Except_All ( :table :GEN_TYPE, :table2 :GEN_TYPE )
         RETURNS :GEN_TYPE ; ,

FUNCTION Intersect ( :table :GEN_TYPE, :table2 :GEN_TYPE )
         RETURNS :GEN_TYPE ; ,

FUNCTION Intersect_All ( :table :GEN_TYPE, :table2 :GEN_TYPE )
         RETURNS :GEN_TYPE ; ,

PRIVATE:

FUNCTION Table_to_Element ( :table :GEN_TYPE ) RETURNS :Element_Type ; ,

FUNCTION Table_to_List ( :table :GEN_TYPE )
         RETURNS SQL_List ( :Element_Type ) ; ,

FUNCTION Empty_Table_to_Table ( :target :GEN_TYPE ) RETURNS :GEN_TYPE ;

) ;
```

Description

1) Let *T* be any abstract or predefined data type. An instance of a generated type SQL_Table(*T*) is a table, each of whose rows is of type *T*.

2) The values of *max_stmt_length*, *max_name_length*, and *max_pred_length* are implementation-dependent maximum lengths for an <SQL procedure statement>, a <descriptor name>, and a <predicate> respectively. *count_type* is the data type of the result of a <set function specification> that simply contains COUNT.

3) Let *F* be an SQL_Insert, SQL_Select, SQL_Update, or SQL_Delete function. Let *T*, *S*, and *D* be the contents of the arguments passed to the *tabref*, *stmt*, and *descr_name* parameters respectively of *F*.

4) If one or more of the following are true, then an exception condition is raised: *syntax error or access rule violation in SQL_Table function*.

   a) Case:

      i) If *F* is an SQL_Insert function and either *S* does not conform to the Format, Syntax Rules, and Access Rules of an <insert statement>, or *S* is an <insert statement> whose immediately contained <table reference> does not identify the same base table as *T*.

      ii) If *F* is an SQL_Select function and either *S* does not conform to the Format, Syntax Rules, and Access Rules of a <query specification>, or *S* is a <query specification> simply containing a <from clause> whose first immediately contained <table reference> does not identify the same base table as *T*.

      iii) If *F* is an SQL_Update function and either *S* does not conform to the Format, Syntax Rules, and Access Rules of an <update statement: searched>, or *S* is an <update statement: searched> whose immediately contained <table reference> does not identify the same base table as *T*.

      iv) If *F* is an SQL_Delete function and either *S* does not conform to the Format, Syntax Rules, and Access Rules of a <delete statement: searched>, or *S* is a <delete statement: searched> whose immediately contained <table reference> does not refer to the same base table as *T*.

   b) *S* contains a parameter reference, an SQL variable reference, or an <embedded variable name>.

5) If *D* identifies an SQL item descriptor area that is not currently allocated, then an exception condition is raised: *invalid SQL descriptor name*.

6) DATA items in dynamic descriptor areas are employed to pass data corresponding to <target specification>s as in Dynamic SQL.

7) When *F* is invoked, all General Rules of the corresponding SQL construct apply to *S*.

8) If *F* is an SQL_Select function, then the result *TQS* of the <query specification> *S* is returned as the result of invocation of *F*.

9) The In function is defined by the Syntax Rules, Access Rules, and General Rules of <in predicate>. The arguments :table and :elem correspond to the evaluated <table subquery> and the <row value constructor> of the <in predicate>, respectively.

10) The Unique function is defined by the Syntax Rules, Access Rules, and General Rules of <unique predicate>. The argument :table corresponds to the evaluated <table subquery> of the <unique predicate>.

11) The Exists function is defined by the Syntax Rules, Access Rules, and General Rules of <exists predicate>. The argument :table corresponds to the evaluated <table subquery> of the <exists predicate>.

12) The For_Some and For_All functions are defined by the Syntax Rules, Access Rules, and General Rules of a <quantified predicate> immediately containing an <existential clause> *EC* or a <universal clause> *UC* respectively. The argument :table corresponds to the first <table reference> of the <table reference list> immediately contained in *EC* or *UC*.

13) The Average, Maximum, Minimum, Sum, and Count functions are defined by the Syntax Rules, Access Rules, and General Rules of the <set function specification>. The argument :table corresponds to the table *TXA* of the <set function specification>.

14) The Distinct function is defined by the Syntax Rules, Access Rules, and General Rules of DISTINCT in a <query specification>.

15) The Union, Union_All, Except, Except_All, Intersect, and Intersect_All functions are defined by the Syntax Rules, Access Rules, and General Rules of the corresponding operators of a <query expression>.

16) If an SQL_Table has only a single element, then the Table_to_Element function returns that element; otherwise an exception condition is raised: *data exception—not singleton collection in cast*.

17) The Table_to_List function returns an SQL_List containing the same elements as the SQL_Table, in an implementation-dependent order.

18) The Empty_Table_to_Table function returns a table of data type SQL_Table(*T*) that is empty, *i.e.*, contains no rows.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no data type generators.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only–SQL3

12.2.4 SQL_Set type template

Function

SQL_Set provides the ability to create and manipulate sets of elements of a specified type.

The SQL_Set type template inherits the functionality of SQL_Table, and applies the constraint that an SQL_Set cannot contain duplicate members. It also defines a subset predicate function.

Definition

```
CREATE TYPE TEMPLATE SQL_Set ( :Element_Type TYPE )
                    UNDER SQL_Table ( :Element_Type )

       (
           !! default constructor and destructor signatures are to be
           !! inserted here when naming problem is resolved

       CAST ( :GEN_TYPE AS :Element_Type WITH Table_to_Element ) ; ,

       CAST ( :GEN_TYPE AS SQL_Table ( :Element_Type ) WITH Set_to_Table ) ; ,

       CAST ( :GEN_TYPE AS SQL_List ( :Element_Type ) WITH Table_to_List ) ; ,

       CAST ( SQL_Empty_Set AS :GENTYPE WITH Empty_Set_to_Set ) ; ,

       CHECK(Unique(VALUE)) ,

       FUNCTION Subset ( :set :GEN_TYPE, :set2 :GEN_TYPE )
                RETURNS BOOLEAN ;

       PRIVATE:

       FUNCTION Set_to_Table ( :set :GEN_TYPE )
                RETURNS SQL_Table ( :Element_Type ) ; ,

       FUNCTION Empty_Set_to_Set ( :target :GEN_TYPE ) RETURNS :GEN_TYPE ;

       ) ;
```

Description

1) Let *T* be any data type. An instance of a generated type SQL_Set(*T*) is a table, each of whose rows is of type *T*.

2) The CHECK condition applies the Unique function, inherited from SQL_Table, to the value of an instance of an SQL_Set (*T*) for some type *T*. This checks for uniqueness of the elements of type *T* of the SQL_Set(*T*).

3) The Subset function returns true if, for every element *E* of the SQL_Set identified by :set, In(:set2, *E*) is true; otherwise it returns false.

4) The Set_to_Table function returns an SQL_Table containing the same elements as the SQL_Set.

5) The Empty_Set_to_Set function returns a set of data type SQL_Set(*T*) that is empty, *i.e.*, contains no rows.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no data type generators.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only–SQL3

12.2.5 SQL_List type template

Function

SQL_List provides the ability to create and manipulate lists of elements of a specified type.

The SQL_List type template inherits the functionality of SQL_Table (redefining several of the functions to take account of ordering in lists), and defines some new functions.

Definition

```
CREATE TYPE TEMPLATE  SQL_List ( :Element_Type TYPE )
                    UNDER SQL_Table ( :Element_Type )

       (

       -- default constructor and destructor signatures are to be inserted here
       -- when naming problem is resolved

       CAST ( :GEN_TYPE AS :Element_Type WITH Table_to_Element ) ; ,

       CAST ( :GEN_TYPE AS SQL_Table ( :Element_Type ) WITH List_to_Table ) ; ,

       CAST (:GEN_TYPE AS SQL_Set ( :Element_Type ) WITH Distinct ) ; ,

       CAST ( SQL_Empty_List AS :GENTYPE WITH Empty_List_to_List ) ; ,

       FUNCTION SQL_Insert ( :tabref REF ( :GEN_TYPE ) ,
                             :stmt CHARACTER VARYING (max_stmt_length) ,
                             :descr_name CHARACTER VARYING (max_name_length) ) ; ,

       FUNCTION SQL_Select ( :tabref REF (:GEN_TYPE ) ,
                             :stmt CHARACTER VARYING (max_stmt_length) ,
                             :descr_name CHARACTER VARYING (max_name_length) )
              RETURNS SQL_Table (Object) ; ,  !! assuming Object is a supertype of all
       ADTs

       FUNCTION SQL_Update ( :tabref REF ( :GEN_TYPE ) ,
                             :stmt CHARACTER VARYING (max_stmt_length) ,
                             :descr_name CHARACTER VARYING (max_name_length) ) ; ,

       FUNCTION SQL_Delete ( :tabref REF ( :GEN_TYPE ) ,
                             :stmt CHARACTER VARYING (max_stmt_length) ,
                             :descr_name CHARACTER VARYING (max_name_length) ) ; ,

       FUNCTION Position ( :list :GEN_TYPE, :elem :Element_Type )
                RETURNS INTEGER ; ,

       FUNCTION Element ( :list :GEN_TYPE, :pos INTEGER )
                RETURNS :Element_Type ; ,

       FUNCTION Head ( :list :GEN_TYPE ) RETURNS :Element_Type ;
                RETURN Element (:list, 1) ; ,
```

```
FUNCTION Tail ( :list :GEN_TYPE ) RETURNS :GEN_TYPE ; ,

FUNCTION Append ( :list :GEN_TYPE, :elem :Element_Type )
         RETURNS :GEN_TYPE ; ,

PRIVATE:

FUNCTION List_to_Table ( :set :GEN_TYPE )
         RETURNS SQL_Table ( :Element_Type ) ; ,

FUNCTION Empty_List_to_List ( :target :GEN_TYPE ) RETURNS :GEN_TYPE ;

 ;
```

Description

1) Let *T* be any data type. An instance of a generated type SQL_List(*T*) is a table, each of whose rows is of type *T*. The rows of an SQL_List have an ordinal position, beginning at 1 for the first element (row).

2) The value of *max_stmt_length* is an implementation-dependent maximum length for an <SQL procedure statement>. The value of *max_name_length* is the maximum length of a <descriptor name>.

3) Let *F* be an SQL_Insert, SQL_Select, SQL_Update, or SQL_Delete function. Let *T*, *S*, and *D* be the contents of the arguments passed to the :tabref, :stmt, and :descr_name parameters respectively of *F*. The behavior of *F*, *T*, *S*, and *D* is as in the description of the SQL_Table type template.

   Note: Note that this does not mean that the behavior is necessarily the same when *T* refers to an SQL_List. The description for an SQL_Table type template depends on the Format, Syntax Rules, Access Rules and General Rules of the statement whose character string representation is the value of the argumentS. The Format and Rules of such a statement differ, depending on whether the statement references an SQL_Table or an SQL_List.

4) The Position function returns the position of the first occurrence of the element specified by :elem in the SQL_List identified by :list, provided the element exists in that list; otherwise, the value 0 is returned.

5) The Element function returns the *i*-th element of the SQL_List identified by :list, where *i* is the value of :pos, provided that such an element exists; otherwise, an exception condition is raised: *SQL_List—invalid element position*.

6) The Head function returns the first element of :list.

7) The Tail function returns an SQL_List(*T*) that is a copy of :list with its first element, if any, removed.

8) The Append function returns an SQL_List(*T*) that is a copy of :list with :elem inserted as the last element.

9) The List_to_Table function returns an SQL_Table containing the same elements as the SQL_List.

10) The Empty_List_to_List function returns a list of data type SQL_List(*T*) that is empty, *i.e.*, contains no rows.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no data type generators.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only–SQL3

12.2.6 SQL_Empty_Table type

Function

SQL_Empty_Table provides the ability to create an empty table, but not to alter its state.

Definition

```
CREATE TYPE SQL_Empty_Table WITHOUT OID
       (
         EQUALS DEFAULT;
         LESS_THAN NONE;

         CONSTRUCTOR FUNCTION SQL_Empty_Table ( ) RETURNS SQL_Empty_Table

       );
```

Description

1) An instance of SQL_Empty_Table is an always empty table, created either by explicit use of its constructor function, or by a <table value expression> containing <table type> TABLE and no <value expressions>s.

2) An instance of SQL_Empty_Table may also be cast to, assigned to, and compared with, an instance of any SQL_Table generated type (see Subclause 12.2.3, "SQL_Table type template").

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <abstract data type>s.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only–SQL3

12.2.7 SQL_Empty_Set type

Function

SQL_Empty_Set provides the ability to create an empty set, but not to alter its state.

Definition

```
CREATE TYPE SQL_Empty_Set WITHOUT OID
           UNDER SQL_Empty_Table
       (
         EQUALS DEFAULT;
         LESS_THAN NONE;

         CONSTRUCTOR FUNCTION SQL_Empty_Set ( ) RETURNS SQL_Empty_Set

       );
```

Description

1) An instance of SQL_Empty_Set is an always empty set, created either by explicit use of its constructor function, or by a <table value expression> containing <table type> SET and no <value expressions>s.

2) An instance of SQL_Empty_Set may also be cast to, assigned to, and compared with, an instance of any SQL_Set generated type (see Subclause 12.2.4, "SQL_Set type template").

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <abstract data type>s.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only–SQL3

12.2.8 SQL_Empty_List type

Function

SQL_Empty_List provides the ability to create an empty list, but not to alter its state.

Definition

```
CREATE TYPE SQL_Empty_List WITHOUT OID
           UNDER SQL_Empty_Table
       (
         EQUALS DEFAULT;
         LESS_THAN NONE;

         CONSTRUCTOR FUNCTION SQL_Empty_List ( ) RETURNS SQL_Empty_List

       );
```

Description

1) An instance of SQL_Empty_List is an always empty list, created either by explicit use of its constructor function, or by a <table value expression> containing <table type> LIST and no <value expressions>s.

2) An instance of SQL_Empty_List may also be cast to, assigned to, and compared with, an instance of any SQL_List generated type (see Subclause 12.2.5, "SQL_List type template").

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <abstract data type>s.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

13 Module

13.1 <module>

Function

Define a module.

Format

```
<module> ::=
       <module name clause>
         <module remainder>
       [ END MODULE ]

<module name clause> ::=
       MODULE [ <module name> ]

<module remainder> ::=
       [ <module character set specification> ]
       <language clause>
       <module authorization clause>
       [ <temporary table declaration>... ]
       <module contents>...

<module character set specification> ::=
       NAMES ARE <character set specification>

<module authorization clause> ::=
         SCHEMA <schema name>
       | AUTHORIZATION <module authorization identifier>
       | SCHEMA <schema name> AUTHORIZATION <module authorization identifier>

<module authorization identifier> ::=
       <authorization identifier>

<module contents> ::=
         <declare cursor>
       | <dynamic declare cursor>
       | <routine>
       | <temporary table declaration>
       | <temporary view definition>
       | <exception declaration>
```

ANSI Only--SQL3

```
       | <SQL variable declaration>
```

Syntax Rules

1) If a <module name clause> does not specify a <module name>, then an implementation-dependent <module name> is implicit.

2) The <module name> shall be different from the <module name> of any other <module> in the same environment.

   Note: An SQL-environment may have multiple <module>s that are unnamed.

3) If the <language clause> of the containing <module> specifies ADA, then a <module name> shall be specified, and that <module name> shall be a valid Ada library unit name.

4) If SCHEMA <schema name> is not specified, then a <schema name> equal to <module authorization identifier> is implicit.

5) If a <module character set specification> is not specified, then a <module character set specification> that specifies an implementation-defined character set that contains at least every character that is in <SQL language character> is implicit.

6) If the explicit or implicit <schema name> does not specify a <catalog name>, then an implementation-defined <catalog name> is implicit.

7) The implicit or explicit <catalog name> is the implicit <catalog name> for all unqualified <schema name>s in the <module>.

8) A <declare cursor> or <dynamic declare cursor> shall precede in the text of the <module> any <routine> that references the <cursor name> of the <declare cursor> or <dynamic declare cursor>.

9) An <exception declaration> shall precede in the text of the module any <exception handler condition> that references the <exception name> of the <exception declaration>.

10) For every <declare cursor> in a <module>, there shall be exactly one <routine> in that <module> that contains an <open statement> that specifies the <cursor name> declared in the <declare cursor>.

    Note: See the Syntax Rules of Subclause 15.1, "<declare cursor>".

ANSI Only–SQL3

11) An <SQL variable declaration> directly contained in <module contents> shall precede in the text of the <module> any <variable specification> that references the <SQL variable name> of the <SQL variable declaration>.

12) An <SQL variable name> contained in a <SQL variable declaration> that is directly contained in <module contents> shall be different from any other <SQL variable name> contained in any other <SQL variable declaration> that is directly contained in <module contents>.

13) A variable declared by an <SQL variable declaration> that is directly contained in <module contents> is referred to as a *module variable*. A module variable is associated with the <module> that immediately contains its <variable declaration>.

Access Rules

None.

General Rules

1) In the SQL-environment, the containing <module> has the name given by the explicit or implicit <module name>.

2) If the SQL-agent performs calls of <routine>s from more than one Ada task, then the results are implementation-dependent.

3) Case:

   a) If a <module authorization identifier> is specified, then it is the current <authorization identifier> for privilege determination for the execution of each <routine> in the <module>.

   b) Otherwise, the current <authorization identifier> for privilege determination for the execution of each <routine> in the <module> is the SQL-session <authorization identifier>.

ANSI Only–SQL3

4) Before the first time that an SQL-agent performs a call of a <procedure> that references the <variable name> of a module variable, that variable is created.

5) After the last time that an SQL-agent performs a call of a <routine>:

   a) A <rollback statement> or a <commit statement> is effectively executed. If an unrecoverable error has occurred, or if the SQL-agent terminated unexpectedly, or if any constraint is not satisfied, then a <rollback statement> is performed. Otherwise, the choice of which of these SQL-statements to perform is implementation-dependent. The determination of whether an SQL-agent has terminated unexpectedly is implementation-dependent.

   b) Let *D* be the <descriptor name> of any system descriptor area that is currently allocated within an SQL-session associated with the SQL-agent. A <deallocate descriptor statement> that specifies

      DEALLOCATE DESCRIPTOR *D*

      is effectively executed.

ANSI Only–SQL3

   c) All module variables associated with this <module> that have been created are destroyed.

   d) All SQL-sessions associated with the SQL-agent are terminated.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <module contents> shall not specify a <temporary view definition>, an <exception declaration>,
   ANSI an <SQL variable declaration>,
   or a <routine> that specifies an external routine.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <module> shall not contain a <temporary table declaration>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <module character set specification> shall not be specified.

   b) A <module> shall be associated with an SQL-agent during its execution. An SQL-agent shall be associated with at most one <module>.

   c) A <module contents> shall not be a <dynamic declare cursor>.

   d) A <module authorization clause> shall specify AUTHORIZATION and shall not specify SCHEMA.

13.2 <routine>

Function

Define a routine.

Format

```
<routine> ::=
       [ CREATE ] <routine header> <routine name> <parameter list>
       [ <returns clause> ]
```

ANSI Only--SQL3

```
       [ <caller language clause> ]
       [ SPECIFIC <specific name> ] <semicolon>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
       [ <caller language clause> ] <semicolon>

       <routine body> <semicolon>

<routine header> ::=
         PROCEDURE
       | [ <function type> ] FUNCTION

<function type> ::=
         CONSTRUCTOR
       | ACTOR
       | DESTRUCTOR

<parameter list> ::=
         <parenthesized style list>
       | <unparenthesized style list>

<parenthesized style list> ::=
       <left paren>
         [ <parameter declaration> [{  <comma> <parameter declaration> }... ] ]
       <right paren>

<unparenthesized style list> ::= [ <parameter declaration>... ]

<parameter declaration> ::=
         <SQL parameter>
       | <external parameter>

<SQL parameter> ::=
         <parameter name> <data type>
       | <status parameter>

<status parameter> ::=
         SQLSTATE
       | SQLCODE

<external parameter> ::= [ <parameter mode> ] <data type>
```

```
<parameter mode> ::=
         IN
       | OUT
       | INOUT

<returns clause> ::= RETURNS <returns data type> [ <result cast> ]

<result cast> ::= CAST FROM <data type>

<returns data type> ::= <data type>

<caller language clause> ::= <language clause>

<routine body> ::=
         <SQL routine body>
       | <external body reference>

<SQL routine body> ::= <SQL procedure statement>

<external body reference> ::=
       EXTERNAL [ NAME <external routine name> ]
       <external routine language clause>
       [ <variant attribute> ]

<variant attribute> ::=
         VARIANT
       | NOT VARIANT

<external routine language clause> ::= <language clause>
```

Syntax Rules

1) A <routine> shall contain a <returns clause> if and only if the <routine header> specifies FUNCTION.

2) A <routine> shall specify CREATE if and only if the <routine> is contained in a <module>.

3) A <routine> that specifies PROCEDURE is called a *procedure*. A <routine> that specifies FUNCTION is called a *function*. A <routine> (procedure or function) whose whose <caller language clause> specifies SQL is called an *SQL-invoked <routine>* (procedure or function). A <routine> (procedure or function) whose <caller language clause> does not specify SQL is called an *externally-invoked <routine>* (procedure or function). A <routine> (procedure or function) that specifies an <SQL body> is called an *SQL <routine>* (procedure or function). A <routine> (procedure or function) that specifies an <external body reference> is called an *external <routine>* (procedure or function).

4) Case:

   a) If a <routine> is contained in a <module>, then it shall not specify a <caller language clause>. A <caller language clause> that is the same as the <language clause> of the <module> is implicit.

   b) If a <routine> is not contained in a <module>, then if a <caller language clause> is not specified, then LANGUAGE SQL is implicit.

5) If the <routine header> of a <routine> specifies a <function type>, then:

   a) The implicit or explicit <caller language clause> shall specify LANGUAGE SQL.

   b) The <routine> shall be an SQL routine.

   c) The <routine> shall be contained in an <abstract data type body> *ADTB*.

      Case:

      i) If *ADTB* is immediately contained in an <explicit abstract data type definition> *ADT*, then let *DT* be the <abstract data type name> immediately contained in *ADT*.

      ii) If *ADTB* is immediately contained in a <type template definition> *TTD*, then let *DT* be ''<GEN_TYPE''.

   d) If the <function type> is CONSTRUCTOR, then:

      i) The <returns data type> shall be *DT*.

      ii) At least one <parameter declaration> shall specify *DT*.

      iii) The <SQL routine body> shall contain exactly one <new statement>, and the operand of that statement shall be a parameter whose data type is *DT*.

   e) If the <function type> is DESTRUCTOR, then:

      i) Exactly one <parameter declaration> shall specify *DT*.

      ii) The <SQL routine body> shall contain exactly one <destroy statement>, and the operand of that statement shall be the parameter whose data type is *DT*.

ANSI Only–SQL3

6) If a <routine> *R* is an SQL-invoked routine that is simply contained in a <schema definition> or <module>, then:

   a) Case:

      i) If *R* is simply contained in a <schema definition>, then let the *subject schema name* be the <schema name> of that <schema definition>.

      ii) If *R* is simply contained in a <module> that is not contained in a <schema>, then let the *subject schema name* be the implicit or explicit <schema name> of the <module name> of that <module> .

   b) If *RN* contains a <local or schema qualifier>, then it shall be the same as the subject <schema name>. If *RN* does not contain a <local or schema qualifier>, then the subject <schema name> is implicit.

   c) The <parameter list> shall not contain a <status parameter>.

   d) The *routine family* of *R* consists of all <routine>s in the subject schema whose <routine name> is equal to *RN* and whose <parameter list> contains the same number of <parameter declaration>s as *R*.

e) If *R* is a procedure (function), then all <routine>s in the <routine family> of *R* shall be procedures (functions). If the *i*-th <parameter declaration> of *R* specifies an input (respectively output, input/output) parameter, then the *i*-th <parameter declaration> of all <routine>s in the routine family of *R* shall specify input (respectively output, input/output) parameters.

f) For each <data type> specified in a <parameter declaration> or <returns clause> that is an abstract data type, let *ADTD* be the <abstract data type definition> that defined the abstract data type. Either *R* shall be contained in *ADTD* or *RN* shall be contained in the <operator name list> of *ADTD*.

7) If a <routine> *R* is an SQL-invoked routine that is simply contained in a <local declaration list> *LDL*, then:

a) If *RN* contains a <local or schema qualifier>, then it shall be the same as the <statement label> of *LDL*. If *RN* does not contain a <local qualifier>, then the <statement label> of *LDL* is implicit.

b) The <parameter list> shall not contain a <status parameter>.

c) The *routine family* of *R* consists of all <routine>s in *LDL* whose <routine name> is equal to *RN* and whose <parameter list> contains the same number of <parameter declaration>s as *R*.

d) If *R* is a procedure (respectively function>, then all <routine>s in the <routine family> of *R* shall be procedures (respectively functions). If the *i*-th <parameter declaration> of R specifies an input (respectively output, input/output) parameter, then the *i*-th <parameter declaration> of all <routine>s in the routine family of *R* shall specify input (respectively output, input/output) parameters.

e) For each <data type> specified in a <parameter declaration> or <returns clause> that is an abstract data type, let *ADTD* be the <abstract data type definition> that defined the abstract data type. Either *R* shall be contained in *ADTD* or *R* shall be contained in a <routine> whose <routine name> is contained in the <operator name list> of *ADTD*.

ISO Only—caused by ANSI changes not yet considered by ISO

---

8) If a <routine> is an SQL-invoked routine, then:

a) The <routine name> shall be a <schema qualified name>.

b) Case:

i) If the <routine> is contained in a <schema definition>, then let the *subject <schema name>* be the <schema name> of that <schema definition>.

ii) If the <routine> is contained in a <module> that is not contained in a <schema>, then let the *subject <schema name>* be the implicit or explicit <schema name> of the <module name> of that <module>.

c) If the <routine name> contains a <schema name>, then that <schema name> shall be the same as the subject <schema name>.

d) If the <routine name> does not contain a <schema name>, then the subject <schema name> is implicit.

ANSI Only–SQL3

e) If <specific name> is not specified, then an implementation-dependent <specific name> is implicit.

****Editor's Note****

Rules like the preceding Rule that require the implementation to "invent" a name for something don't specify what the catalog and schema components of that name should be. This has been identified as a Possible Problem. See Possible Problem **270** in the Editor's Notes.

f) If <specific name> contains a <schema name>, then that <schema name> shall be the same as the subject <schema name>.

g) If <specific name> does not contain a <schema name>, then the subject <schema name> is implicit.

h) The schema identified by the explicit or implicit <schema name> of the <specific name> shall not include a routine descriptor whose instance name is <instance name>.

i) The <parameter list> shall not contain a <status parameter>.

j) The *routine family* for a given <qualified identifer> consists of all <routine>s whose <routine name> is equal to that <qualified identifier>. The *routine family* for a <routine> whose <parameter list> contains *N* <parameter declarations> is all members of the routine family of the <routine name> of the <routine> whose <parameter list>s contain *N* <parameter declarations>.

k) All <routine>s in the routine family for a <routine> that is a procedure shall be procedures. All <routine>s in the routine family of a <routine> that is a function shall be functions. If the *i*-th <parameter declaration> of a <routine> specifies an input (output) parameter, then the *i*-th <parameter declaration> of all <routine>s in the routine family of that <routine> shall specify input (output) parameters.

l) For each <data type> specified in a <parameter declaration> or <returns clause> that is an abstract data type, let *ADTD* be the <abstract data type definition> that defined the abstract data type. Either the <routine> shall be contained in *ADTD* or the <routine name> shall be contained in the <operator name list> of *ADTD*.

9) If a <routine> is an externally-invoked routine, then:

a) The <routine name> shall be an <identifier>.

ANSI Only–SQL3

b) <specific name> shall not be specified.

---

c) If the <routine> is contained in a <module>, then the <identifier> of the <routine name> of the <routine> shall be different from the <identifier> of the <routine name> of any other <routine> in the containing <module>.

d) If the <routine> is contained in a <schema>, then the <identifier> of the <routine name> of the routine shall be different from the <identifier> of the <routine name> of any other <routine> in that <schema>.

e) The <parameter list> shall contain at least one <status parameter>, at most one <status parameter> that specifies SQLCODE, and at most one <status parameter> that specifies SQLSTATE. A parameter that corresponds with SQLCODE is referred to as an SQLCODE parameter. A parameter than corresponds with SQLSTATE is referred to as an SQLSTATE parameter. The SQLCODE and SQLSTATE parameters are referred to as status parameters.

Note: SQLSTATE is the preferred status parameter. The SQLCODE status parameter is a deprecated future that is supported for compatibility with earlier versions of this
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

ANSI Only–SQL3

---

f) No <parameter declaration> shall specify a <data type> that is a table type.

ISO Only—caused by ANSI changes not yet considered by ISO

---

g) No <parameter declaration> shall specify a <data type> that is an object identifier type containing an <abstract data type definition> that implicitly or explicitly specifies WITH OID NOT VISIBLE.

---

h) When the <external routine language clause> specifies ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, then let the *operative data type correspondences table* be Table 17, "Data type correspondences for Ada", Table 18, "Data type correspondences for C", Table 19, "Data type correspondences for COBOL", Table 20, "Data type correspondences for Fortran", Table 21, "Data type correspondences for MUMPS", Table 22, "Data type correspondences for Pascal", or Table 23, "Data type correspondences for PL/I", respectively. Refer to the two columns of the operative data type correspondences table as the ‘‘SQL data type’’ column and the ‘‘host data type column’’.

i) If the <routine> is a function, then let the externally returned data type be the <data type> specified by the <returns data type>.

j) Any <data type> in a <parameter declaration> or externally returned data type (if any) shall specify a data type listed in the SQL data type column for which the corresponding row in the host data type column is not ''none''. If the <data type> in a <parameter declaration> or externally returned data type (if any) specifies the data type listed in the *i*-th row of the SQL data type column, then the type of the *i*-th parameter or the externally returned data type shall be the type listed in the *i*-th row of the host data type column.

k) If the <caller language clause> specifies ADA, then:

i) The <routine>s of the <module> are identified by the <procedure name> as if they were declared immediately within an Ada library unit package specification that has a name identical to the <module name> of the containing <module>.

ii) The Ada base type of any parameter shall be an Ada data type declared in an Ada package named SQL_STANDARD of the following form:

```
package SQL_STANDARD is
       package CHARACTER_SET renames csp;
       subtype CHARACTER_TYPE is CHARACTER_SET.cst;
       type CHAR is array (POSITIVE range <>) of CHARACTER_TYPE;
       type BIT is array (NATURAL range <>) of BOOLEAN;
       type SMALLINT is range bs .. ts;
       type INT is range bi .. ti;
       type REAL is digits dr;
       type DOUBLE_PRECISION is digits dd;
       subtype INDICATOR_TYPE is t;
       type SQLCODE_TYPE is range bsc .. tsc;
       subtype SQL_ERROR is SQLCODE_TYPE range SQL_TYPE'FIRST .. -1;
       subtype NOT_FOUND is SQLCODE_TYPE range 100 .. 100;
       type SQLSTATE_TYPE is new CHAR (1 .. 5);
       package SQLSTATE_CODES is
           AMBIGUOUS_CURSOR_NAME_NO_SUBCLASS:
                   constant SQLSTATE_TYPE := "3C000";
           CASE_NOT_FOUND_FOR_CASE_STATEMENT:
                   constant SQLSTATE_TYPE := "20000";
           CARDINALITY_VIOLATION_NO_SUBCLASS:
                   constant SQLSTATE_TYPE := "21000";
           CONNECTION_EXCEPTION_NO_SUBCLASS:
                   constant SQLSTATE_TYPE := "08000";
           CONNECTION_EXCEPTION_CONNECTION_DOES_NOT_EXIST:
                   constant SQLSTATE_TYPE := "08003";
           CONNECTION_EXCEPTION_CONNECTION_FAILURE:
                   constant SQLSTATE_TYPE := "08006";
           CONNECTION_EXCEPTION_CONNECTION_NAME_IN_USE:
                   constant SQLSTATE_TYPE := "08002";
           CONNECTION_EXCEPTION_SQLCLIENT_UNABLE_TO_ESTABLISH_SQLCONNECTION:

                   constant SQLSTATE_TYPE := "08001";
           CONNECTION_EXCEPTION_SQLSERVER_REJECTED_ESTABLISHMENT_OF_SQLCONNECTION

                   constant SQLSTATE_TYPE := "08004";
           CONNECTION_EXCEPTION_UNKNOWN_STATEMENT_COMPLETION:
                   constant SQLSTATE_TYPE := "08005";
           CURSOR_OPERATION_CONFLICT_NO_SUBCLASS:
                   constant SQLSTATE_TYPE := "09000";
           DATA_EXCEPTION_NO_SUBCLASS:
                   constant SQLSTATE_TYPE := "22000";
           DATA_EXCEPTION_CHARACTER_NOT_IN_REPERTOIRE:
                   constant SQLSTATE_TYPE := "22021";
           DATA_EXCEPTION_DATETIME_FIELD_OVERFLOW:
                   constant SQLSTATE_TYPE := "22008";
           DATA_EXCEPTION_DIVISION_BY_ZERO:
```

```
                constant SQLSTATE_TYPE := "22012";
        DATA_EXCEPTION_ERROR_IN_ASSIGNMENT:
                constant SQLSTATE_TYPE := "22005";
        DATA_EXCEPTION_INDICATOR_OVERFLOW:
                constant SQLSTATE_TYPE := "22022";
        DATA_EXCEPTION_INTERVAL_FIELD_OVERFLOW:
                constant SQLSTATE_TYPE := "22015";
        DATA_EXCEPTION_INVALID_CHARACTER_VALUE_FOR_CAST:
                constant SQLSTATE_TYPE := "22018";
        DATA_EXCEPTION_INVALID_DATETIME_FORMAT:
                constant SQLSTATE_TYPE := "22007";
        DATA_EXCEPTION_INVALID_ESCAPE_CHARACTER:
                constant SQLSTATE_TYPE := "22019";
        DATA_EXCEPTION_INVALID_ESCAPE_SEQUENCE:
                constant SQLSTATE_TYPE := "22025";
        DATA_EXCEPTION_INVALID_FETCH_SEQUENCE:
                constant SQLSTATE_TYPE := "22006";
        DATA_EXCEPTION_INVALID_PARAMETER_VALUE:
                constant SQLSTATE_TYPE := "22023";
```

ANSI Only--SQL3

```
        DATA_EXCEPTION_INVALID_REGULAR_EXPRESSION:
                constant SQLSTATE_TYPE := "2201B";
```

```
        DATA_EXCEPTION_INVALID_TIME_ZONE_DISPLACEMENT_VALUE:
                constant SQLSTATE_TYPE := "22009";
        DATA_EXCEPTION_NULL_VALUE_NO_INDICATOR_PARAMETER:
                constant SQLSTATE_TYPE := "22002";
        DATA_EXCEPTION_NUMERIC_VALUE_OUT_OF_RANGE:
                constant SQLSTATE_TYPE := "22003";
        DATA_EXCEPTION_STRING_DATA_LENGTH_MISMATCH:
                constant SQLSTATE_TYPE := "22026";
        DATA_EXCEPTION_STRING_DATA_RIGHT_TRUNCATION:
                constant SQLSTATE_TYPE := "22001";
        DATA_EXCEPTION_SUBSTRING_ERROR:
                constant SQLSTATE_TYPE := "22011";
        DATA_EXCEPTION_TRIM_ERROR:
                constant SQLSTATE_TYPE := "22027";
        DATA_EXCEPTION_UNTERMINATED_C_STRING:
                constant SQLSTATE_TYPE := "22024";
        DEPENDENT_PRIVILEGE_DESCRIPTORS_STILL_EXIST_NO_SUBCLASS:
                constant SQLSTATE_TYPE := "2B000";
        DYNAMIC_SQL_ERROR_NO_SUBCLASS:
                constant SQLSTATE_TYPE := "07000";
        DYNAMIC_SQL_ERROR_CURSOR_SPECIFICATION_CANNOT_BE_EXECUTED:
                constant SQLSTATE_TYPE := "07003";
        DYNAMIC_SQL_ERROR_INVALID_DESCRIPTOR_COUNT:
                constant SQLSTATE_TYPE := "07008";
        DYNAMIC_SQL_ERROR_INVALID_DESCRIPTOR_INDEX:
                constant SQLSTATE_TYPE := "07009";
        DYNAMIC_SQL_ERROR_PREPARED_STATEMENT_NOT_A_CURSOR_SPECIFICATION:
                constant SQLSTATE_TYPE := "07005";
        DYNAMIC_SQL_ERROR_RESTRICTED_DATA_TYPE_ATTRIBUTE_VIOLATION:
                constant SQLSTATE_TYPE := "07006";
        DYNAMIC_SQL_ERROR_USING_CLAUSE_DOES_NOT_MATCH_DYNAMIC_PARAMETER_SPEC:

                constant SQLSTATE_TYPE := "07001";
        DYNAMIC_SQL_ERROR_USING_CLAUSE_DOES_NOT_MATCH_TARGET_SPEC:
                constant SQLSTATE_TYPE := "07002";
        DYNAMIC_SQL_ERROR_USING_CLAUSE_REQUIRED_FOR_DYNAMIC_PARAMETERS:
                constant SQLSTATE_TYPE := "07004";
```

```
DYNAMIC_SQL_ERROR_USING_CLAUSE_REQUIRED_FOR_RESULT_FIELDS:
        constant SQLSTATE_TYPE := "07007";
EXTERNAL FUNCTION CALL EXCEPTION:
        constant SQLSTATE_TYPE := "39000";
```

ANSI Only--SQL3

```
EXTERNAL FUNCTION CALL EXCEPTION_INVALID_SQLSTATE_RETURNED:
        constant SQLSTATE_TYPE := "39001";
```

```
EXTERNAL FUNCTION EXCEPTION:
        constant SQLSTATE_TYPE := "38000";
EXTERNAL FUNCTION EXCEPTION_INVALID_RETURN_VALUE_FOR_PREDICATE:
        constant SQLSTATE_TYPE := "39002";
EXTERNAL FUNCTION EXCEPTION_NOT_AVAILABLE:
        constant SQLSTATE_TYPE := "39001";
FEATURE_NOT_SUPPORTED_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "0A000";
FEATURE_NOT_SUPPORTED_MULTIPLE_ENVIRONMENT_TRANSACTIONS:
        constant SQLSTATE_TYPE := "0A001";
INTEGRITY_CONSTRAINT_VIOLATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "23000";
INVALID_AUTHORIZATION_SPECIFICATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "28000";
INVALID_CATALOG_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "3D000";
INVALID_CHARACTER_SET_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "2C000";
INVALID_CONDITION_NUMBER_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "35000";
INVALID_CONNECTION_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "2E000";
INVALID_CURSOR_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "34000";
INVALID_CURSOR_STATE_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "24000";
INVALID_SCHEMA_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "3F000";
INVALID_SQL_DESCRIPTOR_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "33000";
INVALID_SQL_STATEMENT_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "26000";
INVALID_TRANSACTION_STATE_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "25000";
INVALID_TRANSACTION_TERMINATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "2D000";
NO_DATA_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "02000";
REMOTE_DATABASE_ACCESS_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "HZ000";
SUCCESSFUL_COMPLETION_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "00000";
SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "42000";
SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION_IN_DIRECT_STATEMENT_NO_SUBCLASS

        constant SQLSTATE_TYPE := "2A000";
SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION_IN_DYNAMIC_STATEMENT_NO_SUBCLASS

        constant SQLSTATE_TYPE := "37000";
TRANSACTION_ROLLBACK_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "40000";
```

```
                TRANSACTION_ROLLBACK_INTEGRITY_CONSTRAINT_VIOLATION:
                        constant SQLSTATE_TYPE := "40002";
                TRANSACTION_ROLLBACK_SERIALIZATION_FAILURE:
                        constant SQLSTATE_TYPE := "40001";
                TRIGGERED_DATA_CHANGE_VIOLATION_NO_SUBCLASS:
                        constant SQLSTATE_TYPE := "27000";
                WARNING_NO_SUBCLASS:
                        constant SQLSTATE_TYPE := "01000";
                WARNING_CONSTRAINT_CURSOR_OPERATION_CONFLICT:
                        constant SQLSTATE_TYPE := "01001";
                WARNING_DISCONNECT_ERROR:
                        constant SQLSTATE_TYPE := "01002";
```

ANSI Only--SQL3

```
                WARNING_EXTERNAL_ROUTINE_WARNING:
                        constant SQLSTATE_TYPE := "01H00";
```

```
                WARNING_IMPLICIT_ZERO_BIT_PADDING:
                        constant SQLSTATE_TYPE := "01008";
                WARNING_INSUFFICIENT_ITEM_DESCRIPTOR_AREAS:
                        constant SQLSTATE_TYPE := "01005";
                WARNING_NULL_VALUE_ELIMINATED_IN_SET_FUNCTION:
                        constant SQLSTATE_TYPE := "01003";
                WARNING_PRIVILEGE_NOT_GRANTED:
                        constant SQLSTATE_TYPE := "01007";
                WARNING_PRIVILEGE_NOT_REVOKED:
                        constant SQLSTATE_TYPE := "01006";
                WARNING_SEARCH_CONDITION_TOO_LONG_FOR_INFORMATION_SCHEMA:
                        constant SQLSTATE_TYPE := "01009";
                WARNING_STRING_DATA_RIGHT_TRUNCATION_WARNING:
                        constant SQLSTATE_TYPE := "01004";
                WARNING_QUERY_EXPRESSION_TOO_LONG_FOR_INFORMATION_SCHEMA:
                        constant SQLSTATE_TYPE := "0100A";
                WITH_CHECK_OPTION_VIOLATION_NO_SUBCLASS:
                        constant SQLSTATE_TYPE := "44000";
        end SQLSTATE_CODES;
    end SQL_STANDARD;
```

where *csp* is an implementation-defined package and *cst* is an implementation-defined character type such that within the scope of an Ada **use** clause for SQL_STANDARD.CHARACTER_SET, string literals can be of type SQL_STANDARD.CHAR. *bs*, *ts*, *bi*, *ti*, *dr*, *dd*, *bsc*, and *tsc* are implementation-defined integer values. *t* is INT or SMALLINT, corresponding with an implementation-defined <exact numeric type> of indicator parameters.

SQL_STANDARD shall contain no other declarations.

10) If a <routine> is an SQL routine, then:

a) Each <parameter declaration> in the <parameter list> shall be an <SQL parameter>.

b) The <parameter name> of each <SQL parameter> in a <parameter list> shall be different from the <parameter name> of any other <SQL parameter> in that <parameter list>.

c) Whether a <parameter declaration> is for an *input parameter*, an *output parameter*, or both is determined as follows:

Case:

i) A <status parameter> is an *output parameter*.

ii) For every <parameter declaration> that is not a <status parameter>,

Case:

1) If the <parameter name> of a parameter is contained in a <value specification> or a <simple value specification> that is contained in <SQL procedure statement>, but it is not contained in a <target specification> or a <simple target specification> that is contained in <SQL procedure statement>, then the parameter is an *input parameter*.

2) If the <parameter name> of a parameter is contained in a <target specification> or a <simple target specification> that is contained in <SQL procedure statement>, but it is not contained in a <value specification> or a <simple value specification> that is contained in <SQL procedure statement>, then the parameter is an *output parameter*.

3) If the <parameter name> of a parameter is contained in a <value specification> or a <simple value specification> that is contained in <SQL procedure statement> and it is contained in a <target specification> or a <simple target specification> that is contained in <SQL procedure statement>, then the parameter is both an *input parameter* and an *output parameter*.

4) Otherwise, the parameter is neither an *input parameter* nor an *output parameter*.

d) Any <parameter name> contained in the <SQL routine body> of a <routine> shall be specified in an <SQL parameter> of the <routine>.

e) If the <SQL routine body> contains an <enumeration literal> that is identical to a <parameter name> in the <parameter list>, then the <enumeration literal> shall contain a <schema name>.

f) The <returns clause> shall not specify a <result cast>.

g) An <SQL routine body> shall contain a <return statement> if and only if the <routine> is a function.

11) If a <routine> is an external routine, then:

a) Each <parameter declaration> in the <parameter list> shall be an <external parameter>.

b) The implicit or explicit <caller language clause> shall specify SQL.

c) If an <external routine name> is not specified, then the <qualified identifier> of the <routine name> is implicit.

d) If a <variant attribute> is not specified, then VARIANT is implicit.

e) The <external routine language clause> shall not specify SQL.

f) If a <result cast> is specified, then let *R* be some value of the <data type> specified in the <result cast> and let *RT* be the <returns data type>. The following shall be valid according to the Syntax Rules of Subclause 6.13, "<cast specification>":

CAST (*R* AS *RT*)

g) If <parameter mode> is not specified, then IN is implicit.

h) If <parameter mode> is IN or INOUT, then the parameter is an *input parameter*. If the <parameter mode> is OUT or INOUT, then the parameter is an *output parameter*.

i) Let *N* be the number of <external parameter>s. Let *PN* be the number of <external parameter>s that are predefined types. Let *UN* be the number of actual base types of the operand data types that are abstract data types. If the <returns data type> is an abstract data type, then let *RN* be the number of actual base types of that data type; otherwise, let *RN* be 1. Let the *effective external parameter type list* be a list of $PN + UN + RN + N + 2$ data types, as follows:

i) Effective parameter type list entries 1 to $PN+UN$ are *N* groups of entries. For *i* ranging from 1 to *N*,

Case:

1) If the *i*-th <external parameter> is a predefined type, then the *i*-th group of effective parameter type list entries consists of the *i*-th <external parameter>.

2) If the *i*-th <external parameter> is some abstract data type ADT_i, then the *i*-th group of effective parameter type list entries consists of the <parameter mode> of the <external parameter> and the actual base types of the <abstract data type definition> of ADT_i.

ii) Effective parameter type list entries $(PN + UN) + 1$ to $(PN + UN) + RN$ are as follows:

Case:

1) If the <returns data type> is a predefined type, then effective external parameter list entry $(PN + UN) + 1$ is the <parameter mode> OUT and the <returns data type>.

2) If the <returns data type> is some abstract data type ADT_r, then effective external parameter list entries (PN+UN)+1 to $(PN + UN) + RN$ consist of the <parameter mode> OUT and the actual base types of the <abstract data type definition> of ADT_r.

iii) Effective external parameter list entries $(PN+UN+RN)+1$ to $(PN+UN+RN)+N+1$ are $N + 1$ occurrences of an implementor-defined <data type> that is an exact numeric type with scale 0. The <parameter mode> for the *i*-th such effective external parameter is the same as that of the $i - RN - PN - UN$-th effective external parameter.

iv) Effective parameter type list entry $(PN + UN + RN + N + 1) + 1$ is a <data type> that is character string of length
ANSI 5
ISO 3
and character set SQL_TEXT with <parameter mode>
ANSI INOUT. This parameter is used to pass the exception data item.
ISO OUT.

j) Depending on whether the <caller language clause> specifies ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, let the *operative data type correspondences table* be Table 17, "Data type correspondences for Ada", Table 18, "Data type correspondences for C", Table 19, "Data type correspondences for COBOL", Table 20, "Data type correspondences for Fortran", Table 21, "Data type correspondences for MUMPS", Table 22, "Data type correspondences

for Pascal", or Table 23, "Data type correspondences for PL/I", respectively. Refer to the two columns of the operative data type correspondences table as the ''SQL data type'' column and the ''host data type column''.

k) If the <routine> is a function, then if the <routine> contains a <result cast>, let the externally returned data type be the <data type> contained in that <result cast>; otherwise, let the externally returned data type be the <data type> contained in the <returns clause>.

ANSI Only–SQL3

l) Any <data type> in a <parameter declaration> or externally returned data type (if any) shall not specify a table type.

m) Any <data type> in a <parameter declaration> or externally returned data type (if any) shall specify a data type listed in the SQL data type column for which the corresponding row in the host data type column is not ''none''. If the <data type> in a <parameter declaration> or externally returned data type (if any) specifies the data type listed in the *i*-th row of the SQL data type column, then the type of the *i*-th parameter or the externally returned data type shall be the type listed in the *i*-th row of the host data type column.

Access Rules

1) If an external routine is contained in a <module>, then the <user authorization identifier> of the current SQL-session shall be equal to the <user authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <routine>.

General Rules

1) If the <routine> is an external routine, then a privilege descriptor is created that defines the EXTERNAL PRIVILEGES privilege on the <routine> to the <authorization identifier> of the schema or <module> in which the <routine> appears. This privilege is grantable. The grantor for the privilege descriptor is set to the special grantor value ''_SYSTEM''.

ANSI Only–SQL3

2) If the <routine> is an external routine, then the method and time of binding of that external routine to the schema or <module> containing the <routine> is implementation-defined. If the designated external routine is not a program that conforms to the programming language standard specified by the <external routine language clause>, then the results are implementation-dependent.

3) Case:

a) If a <routine> is an SQL-invoked routine, then the rules for invocation of the <routine> are specified in Subclause 10.3, "<routine invocation>".

b) If a <routine> is an externally-invoked routine, then the rules for invocation of the <routine> are specified in Subclause 13.5, "Rules for externally-invoked <routine>s".

ANSI Only–SQL3

4) A routine descriptor is created that describes the routine being defined:

a) The routine name included in the routine descriptor is <routine name>.

b) If the routine is an SQL-invoked routine, then the routine descriptor includes the specific name.

c) The routine descriptor includes, for each parameter in <parameter list>, the name, data type, position, and an indication of whether the parameter is input, output, or both.

d) If the routine is a function, then the routine descriptor includes the data type in the <returns data type>.

e) If the routine is defined within an abstract data type definition or a table definition, then the routine descriptor includes an indication of its encapsulation level.

f) If the routine is an SQL-invoked routine that is an SQL routine, then the function type of the routine if <function type>.

g) The name of the caller language of the routine is the <language name> in the <language clause> directly contained in <caller language clause>.

h) If the routine is an external routine, then the external name of the routine is <external routine name>.

i) If the routine is an external routine, then the name of the language in which the body of the routine was written is the <language name> in the <language clause> directly contained in <external routine language clause>.

j) If the routine is an external routine, then the routine descriptor includes an indication of whether the routine is VARIANT or NOT VARIANT.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <routine> shall be a routine, shall be contained in a <module>, and shall not contain a <caller language clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

13.3 <SQL procedure statement>

Function

Define all of the SQL-statements that are <SQL procedure statement>s.

Format

```
<SQL procedure statement> ::=
       [ ASYNC <left paren> <async statement identifier> <right paren> ]
            <SQL executable statement>

<async statement identifier> ::= <numeric value expression>

<SQL executable statement> ::=
         <SQL schema statement>
       | <SQL control statement>
       | <SQL data statement>
       | <SQL transaction statement>
       | <SQL connection statement>
       | <SQL session statement>
       | <SQL dynamic statement>
       | <SQL diagnostics statement>

<SQL schema statement> ::=
         <SQL schema definition statement>
       | <SQL schema manipulation statement>

<SQL schema definition statement> ::=
         <schema definition>
       | <table definition>
       | <view definition>
       | <grant statement>
       | <role definition>
       | <grant role statement>
       | <domain definition>
       | <null class definition>
       | <character set definition>
       | <collation definition>
       | <translation definition>
       | <assertion definition>
       | <trigger definition>
       | <routine>
       | <abstract data type definition>
       | <type template definition>
       | <module definition>

<SQL schema manipulation statement> ::=
         <drop schema statement>
       | <alter table statement>
       | <drop table statement>
       | <drop view statement>
       | <revoke statement>
       | <revoke role statement>
       | <drop role statement>
       | <alter domain statement>
       | <drop domain statement>
       | <drop null class statement>
       | <drop character set statement>
       | <drop collation statement>
```

```
        | <drop translation statement>
        | <drop assertion statement>
        | <drop trigger statement>
        | <drop routine statement>
        | <drop data type statement>
        | <drop type template statement>
        | <drop module statement>

<SQL control statement> ::=
          <call statement>
        | <return statement>
        | <assignment statement>
        | <compound statement>
        | <new statement>
        | <destroy statement>
        | <case statement>
        | <if statement>
        | <leave statement>
        | <loop statement>
```

ANSI Only--SQL3

```
        | <for statement>
```

```
<SQL data statement> ::=
          <open statement>
        | <fetch statement>
        | <close statement>
        | <select statement: single row>
        | <SQL data change statement>

<SQL data change statement> ::=
          <delete statement: positioned>
        | <delete statement: searched>
        | <insert statement>
        | <update statement: positioned>
        | <update statement: searched>

<SQL transaction statement> ::=
          <set transaction statement>
        | <set constraints mode statement>
        | <test completion statement>
        | <savepoint statement>
        | <release savepoint statement>
        | <commit statement>
        | <rollback statement>

<SQL connection statement> ::=
          <connect statement>
        | <set connection statement>
        | <disconnect statement>

<SQL session statement> ::=
          <set catalog statement>
        | <set schema statement>
        | <set names statement>
        | <set session authorization identifier statement>
        | <set role statement>
        | <set local time zone statement>
```

```
<SQL dynamic statement> ::=
          <system descriptor statement>
        | <prepare statement>
        | <deallocate prepared statement>
        | <describe statement>
        | <execute statement>
        | <execute immediate statement>
        | <SQL dynamic data statement>

<SQL dynamic data statement> ::=
          <allocate cursor statement>
        | <dynamic open statement>
        | <dynamic fetch statement>
        | <dynamic close statement>
        | <dynamic delete statement: positioned>
        | <dynamic update statement: positioned>

<system descriptor statement> ::=
          <allocate descriptor statement>
        | <deallocate descriptor statement>
        | <set descriptor statement>
        | <get descriptor statement>

<SQL diagnostics statement> ::=
          <get diagnostics statement>
        | <signal statement>
        | <resignal statement>
```

Syntax Rules

- 1 Rule deleted.

None.

Access Rules

None.

General Rules

1) If <SQL procedure statement> is not a <compound statement>, then:

   a) If any exception condition is raised during the execution of <SQL procedure statement> *S*, either by *S* or any triggered <SQL procedure statement>, then one of the exception conditions is the active condition.

   b) All changes made to SQL-data or schemas by the execution of *S* and any <SQL procedure statement>s triggered by the execution of *S* are canceled.

   c) Diagnostics information resulting from the execution of *S* is placed into the diagnostics area as specified in Subclause 20.1, "<get diagnostics statement>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <SQL procedure statement> shall not specify ASYNC.

   b) An <SQL executable statement> shall not be a <SQL control statement>.

   c) An <SQL schema definition statement> shall not be a <role definition>, <grant role statement>, <null class definition>, <trigger definition>, <routine>, <abstract data type definition>, <type template definition>, or <module definition>.

   d) An <SQL shema manipulation statement> shall not be a <revoke role statement>, <drop role statement>, <drop null class statement>, <drop trigger statement>, <drop routine statement>, <drop data type statement>, <drop type template statement>, or <drop module statement>.

   e) An <SQL transaction statement> shall not be a <test completion statement>, <savepoint statement>, or <release savepoint statement>.

   f) An <SQL session statement> shall not be a <set role statement>.

   g) An <SQL diagnostics statement> shall not be a <signal statement> or <resignal statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) An <SQL procedure statement> shall not be an <SQL schema definition statement>.

13.4 Data type correspondences

Function

Specify the data type correspondences for SQL data types and host language types.

Note: These tables are referenced in Subclause 13.2, "<routine>", for the definitions of externally-invoked routines and external routines.

In the following tables, let *P* be <precision>, *S* be <scale>, *L* be <length>, *T* be <time fractional seconds precision>, and *Q* be <interval qualifier>.

Tables

Table 17—Data type correspondences for Ada

ISO Only—caused by ANSI changes not yet considered by ISO

SQL Data Type	Ada Data Type
SQLSTATE	SQL_STANDARD.SQLSTATE_TYPE
SQLCODE	SQL_STANDARD.SQLCODE_TYPE
CHARACTER(*L*)	SQL_STANDARD.CHAR, with P'LENGTH of *L*
CHARACTER VARYING(*L*)	*None*
BIT(*L*)	SQL_STANDARD.BIT, with P'LENGTH of *L*
BIT VARYING(*L*)	*None*
BOOLEAN	SQL_STANDARD.BOOLEAN
SMALLINT	SQL_STANDARD.SMALLINT
INTEGER	SQL_STANDARD.INT
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	*None*
REAL	SQL_STANDARD.REAL
DOUBLE PRECISION	SQL_STANDARD.DOUBLE_PRECISION
FLOAT(*P*)	*None*
DATE	*None*
TIME*T*	*None*
TIMESTAMP*T*	*None*
INTERVAL*Q*	*None*

ANSI Only–SQL3

Table 17—Data type correspondences for Ada (Cont.)

SQL Data Type	Ada Data Type
object identifier type	*None*
ISO Only—caused by ANSI changes not yet considered by ISO	
object ADT	SQL_STANDARD.CHAR, with P'LENGTH of *X*[1]

[1]The length *X* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.

Table 18—Data type correspondences for C

ISO Only—caused by ANSI changes not yet considered by ISO

SQL Data Type	C Data Type
SQLCODE	pointer to long
SQLSTATE	char, with length 6
CHARACTER(*L*)	char, with length *L*+1
CHARACTER VARYING (*L*)	char, with length *L*+1
BIT(*L*)	char, with length *X*[1]
BIT VARYING(*L*)	*None*
BOOLEAN	pointer to long
SMALLINT	pointer to short
INTEGER	pointer to long
DECIMAL(*P*,*S*)	None
NUMERIC(*P*,*S*)	None
REAL	pointer to float
DOUBLE PRECISION	pointer to double
FLOAT(*P*)	*None*
DATE	*None*
TIME*T*	*None*
TIMESTAMP*T*	*None*
INTERVAL*Q*	*None*

ANSI Only–SQL3

object identifier type	*None*

ISO Only—caused by ANSI changes not yet considered by ISO

object ADT	char, with length *Y*[2]

[1]The length *X* of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in character of the host language.

Table 18—Data type correspondences for C (Cont.)

SQL Data Type	C Data Type
[2]The length *Y* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.	

Table 19—Data type correspondences for COBOL

ISO Only—caused by ANSI changes not yet considered by ISO

SQL Data Type	COBOL Data Type
SQLSTATE	PICTURE X(5)
SQLCODE	PICTURE S9(*PC*) USAGE COMPUTATIONAL, where *PC* is implementation-defined between 4 and 18, inclusive
CHARACTER(*L*)	alphanumeric, with length *L*
CHARACTER VARYING(L)	*None*
BIT(*L*)	alphanumeric, with length *X*[1]
BIT VARYING(*L*)	*None*
BOOLEAN	PICTURE X
SMALLINT	PICTURE S9(*SPI*) USAGE BINARY, where *SPI* is implementation-defined
INTEGER	PICTURE S9(*PI*) USAGE BINARY, where *PI* is implementation-defined
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	USAGE DISPLAY SIGN LEADING SEPARATE, with PICTURE as specified[2]
REAL	*None*
DOUBLE PRECISION	*None*
FLOAT(*P*)	*None*
DATE	*None*
TIME*T*	*None*
TIMESTAMP*T*	*None*
INTERVAL*Q*	*None*

ANSI Only–SQL3

object identifier type	*None*

ISO Only—caused by ANSI changes not yet considered by ISO

[1]The length of a character type corresponding with SQL BIT(*L*) is one more than the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in one character of the host language.

Table 19—Data type correspondences for COBOL (Cont.)

SQL Data Type	COBOL Data Type
[2]Case: a) If *S*=*P*, then a PICTURE with an 'S' followed by a 'V' followed by *P* '9's. b) If *P*>*S*>0, then a PICTURE with an 'S' followed by *P*–*S* '9's followed by a 'V' followed by *S* '9's. c) If *S*=0, then a PICTURE with an 'S' followed by *P* '9's optionally followed by a 'V'.	

Table 19—Data type correspondences for COBOL (Cont.)

SQL Data Type	COBOL Data Type
object ADT	alphanumeric, with length *Y*[3]

[3]The length *Y* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.

Table 20—Data type correspondences for Fortran

ISO Only—caused by ANSI changes not yet considered by ISO

SQL Data Type	Fortran Data Type
SQLSTATE	CHARACTER, with length 5
SQLCODE	INTEGER
CHARACTER(*L*)	CHARACTER, with length *L*
CHARACTER VARYING(*L*)	*None*
BIT(*L*)	CHARACTER, with length *X*[1]
BIT VARYING(*L*)	*None*
BOOLEAN	LOGICAL
SMALLINT	*None*
INTEGER	INTEGER
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	*None*
REAL	REAL
DOUBLE PRECISION	DOUBLE PRECISION
FLOAT(*P*)	*None*
DATE	*None*
TIME*T*	*None*
TIMESTAMP*T*	*None*
INTERVAL*Q*	*None*

ANSI Only–SQL3

object identifier type	*None*

ISO Only—caused by ANSI changes not yet considered by ISO

object ADT	CHARACTER, with length *Y*[2]

[1]The length *X* of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in character of the host language.

Table 20—Data type correspondences for Fortran (Cont.)

SQL Data Type	Fortran Data Type
[2]The length *Y* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.	

Table 21—Data type correspondences for MUMPS

ISO Only—caused by ANSI changes not yet considered by ISO

SQL Data Type	MUMPS Data Type
SQLSTATE	character, with maximum length at least 5
SQLCODE	*None*
CHARACTER(*L*)	*None*
CHARACTER VARYING (*L*)	character with maximum length *L*
BIT(*L*)	*None*
BIT VARYING(*L*)	*None*
BOOLEAN	*None*
SMALLINT	*None*
INTEGER	character
DECIMAL(*P*,*S*)	character
NUMERIC(*P*,*S*)	character
REAL	character
DOUBLE PRECISION	*None*
FLOAT(*P*)	*None*
DATE	*None*
TIME*T*	*None*
TIMESTAMP*T*	*None*
INTERVAL*Q*	*None*

ANSI Only–SQL3

object identifier type	*None*

ISO Only—caused by ANSI changes not yet considered by ISO

object ADT	CHARACTER, with length *X*[1]

[1]The length *X* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.

Table 22—Data type correspondences for Pascal

ISO Only—caused by ANSI changes not yet considered by ISO

SQL Data Type	Pascal Data Type
SQLSTATE	PACKED ARRAY[1..5] OF CHAR
SQLCODE	INTEGER
CHARACTER(1)	CHAR
CHARACTER(*L*), *L*>1	PACKED ARRAY[1..*L*] OF CHAR
CHARACTER VARYING(L)	*None*
BIT(*L*), 1 $\leq L \leq B^1$	CHAR
BIT(*L*), $B^1 < L$	PACKED ARRAY[LB[1]] OF CHAR
BIT VARYING(*L*)	*None*
BOOLEAN	BOOLEAN
SMALLINT	*None*
INTEGER	INTEGER
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	*None*
REAL	REAL
DOUBLE PRECISION	*None*
FLOAT(*P*)	*None*
DATE	*None*
TIME*T*	*None*
TIMESTAMP*T*	*None*
INTERVAL*Q*	*None*

ANSI Only–SQL3

object identifier type	*None*

ISO Only—caused by ANSI changes not yet considered by ISO

[1]The length *LB* of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in a character of the host language.

(Continued on next page)

Table 22—Data type correspondences for Pascal (Cont.)

SQL Data Type	Pascal Data Type
object ADT	CHAR, with length X^2

[2]The length *X* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.

Table 23—Data type correspondences for PL/I

ISO Only—caused by ANSI changes not yet considered by ISO

SQL Data Type	PL/I Data Type
SQLSTATE	CHARACTER(5)
SQLCODE	FIXED BINARY(*PP*), where *PP* is an implementation-defined precision at least 15.
CHARACTER(*L*)	CHARACTER(*L*)
CHARACTER VARYING(*L*)	CHARACTER VARYING(*L*)
BIT(*L*)	BIT(*L*)
BIT VARYING(*L*)	BIT VARYING (*L*)
BOOLEAN	BIT(1)
SMALLINT	FIXED BINARY(*SPI*), where *SPI* is implementation-defined
INTEGER	FIXED BINARY(*PI*), where *PI* is implementation-defined
DECIMAL(*P*,*S*)	FIXED REAL DECIMAL(*P*,*S*)
NUMERIC(*P*,*S*)	*None*
REAL	*None*
DOUBLE PRECISION	*None*
FLOAT(*P*)	FLOAT(*P*)
DATE	*None*
TIME*T*	*None*
TIMESTAMP*T*	*None*
INTERVAL*Q*	*None*

ANSI Only–SQL3

object identifier type	*None*

ISO Only—caused by ANSI changes not yet considered by ISO

object ADT	CHARACTER, with length *X*[1]

[1]The length *X* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.

13.5 Rules for externally-invoked <routine>s

Function

Specify General Rules for externally-invoked <routine>s.

General Rules

1) A <routine> defines a procedure or function that may be called by an SQL-agent.

2) If a <routine> is contained in a <module> that is associated with an SQL-agent that is associated with another <module> that contains a <routine> with the same <routine name>, then the effect is implementation-defined.

3) If the SQL-agent that performs a call of a <routine> is not a program that conforms to the programming language standard specified by the implicit or explicit <caller language clause> of that <routine>, then the effect is implementation-dependent.

4) If the implicit or explicit <caller language clause> of a <routine> specifies ADA and the SQL-agent performs calls of <routine>s from more than one Ada task, then the results are implementation-dependent.

5) If the <module> that contains the <routine> has an explicit <module authorization identifier> *MAI* that is different from the SQL-session <authorization identifier> *SAI*, then:

   a) Whether or not *SAI* can invoke <routine>s in a <module> with explicit <module authorization identifier> *MAI* is implementation-defined, as are any restrictions pertaining to such invocation.

   b) If *SAI* is restricted from invoking a <routine> in a <module> with explicit <module authorization identifier> *MAI*, then an exception condition is raised: *invalid authorization specification*.

6) If the value of any input parameter provided by the SQL-agent falls outside the set of allowed values of the data type of the parameter, or if the value of any output parameter resulting from the execution of the <routine> falls outside the set of values supported by the SQL-agent for that parameter, then the effect is implementation-defined. If the implementation-defined effect is the raising of an exception condition, then an exception condition is raised: *data exception—invalid parameter value*.

ISO Only—caused by ANSI changes not yet considered by ISO

7) If an input parameter is an object abstract data type, then the value provided by the standard programming language with the actual parameter in the procedure call is a character string that is interpreted as an object identifier, and the input parameter is bound to the object identified by that object identifier. If the object identifier does not identify an existing object of the specified object abstract data type, then an exception condition is raised: *data exception–invalid parameter value*.

8) If an output parameter is an object abstract data type, then the object identifier of the object referenced by that parameter is materialized as a character string and is assigned to the actual parameter in the procedure call.

9) Let *S* be the <SQL procedure statement> of the <routine>.

10) When the <routine> is called by an SQL-agent:

Case:

a) If ASYNC is specified and there are already an implementation-defined number of outstanding asynchronous <SQL procedure statement>s and *S* is not a <test completion statement> or a <rollback statement>, then an exception condition is raised: *asynchronous SQL statement not accepted*.

b) If ASYNC is specified and the <async statement identifier> is equal to any <async statement identifier> of any <SQL procedure statement> initiated for asynchronous execution in the same SQL-transaction, the an exception condition is raised: *duplicate asynchronous SQL statement identifier*.

c) If *S* is an <SQL connection statement>, then:

i) The <module> that contains *S* is associated with the SQL-agent.

ii) The diagnostics area is emptied.

iii) *S* is effectively executed.

iv) If *S* successfully initiated or resumed an SQL-session, then subsequent calls to a <routine> by the SQL-agent are associated with that SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

d) If *S* is an <SQL diagnostics statement>, then:

i) The <module> that contains *S* is associated with the SQL-agent.

ii) *S* is executed.

e) Otherwise:

i) If no SQL-session is current for the SQL-agent, then

Case:

1) If the SQL-agent has not executed an <SQL connection statement> and there is no default SQL-session associated with the SQL-agent, then the following <connect statement> is executed:

CONNECT TO DEFAULT

2) If the SQL-agent has not executed an <SQL connection statement> and there is a default SQL-session associated with the SQL-agent, then the following <set connection statement> is effectively executed:

SET CONNECTION DEFAULT

3) Otherwise, an exception condition is raised: *connection exception—connection does not exist*.

Subsequent calls to a <routine> or invocations of <direct SQL statement>s by the SQL-agent are associated with the SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

ii) If an SQL-transaction is active for the SQL-agent, then *S* is associated with that SQL-transaction.

iii) If no SQL-transaction is active for the SQL-agent and *S* is a transaction-initiating SQL-statement, then

1) An SQL-transaction is effectively initiated and associated with this call and with subsequent calls of any <routine> or invocations of <direct SQL statement>s by that SQL-agent until the SQL-agent terminates that SQL-transaction.

2) Case:

A) If a <set transaction statement> has been executed since the termination of the last SQL-transaction in the SQL-session, then the access mode, constraint mode, and isolation level of the SQL-transaction are set as specified by the <set transaction statement>.

B) Otherwise, the access mode of that SQL-transaction is *read-write*, the constraint mode for all constraints in that SQL-transaction is *immediate*, and the isolation level of that SQL-transaction is SERIALIZABLE.

3) The SQL-transaction is associated with the SQL-session.

4) The <module> that contains *S* is associated with the SQL-transaction.

iv) The <module> that contains *S* is associated with the SQL-agent.

v) If *S* contains an <SQL schema statement> and the access mode of the current SQL-transaction is *read-only*, then an exception condition is raised: *invalid transaction state*.

vi) The diagnostics area is emptied.

vii) The values of all input parameters to the <routine> are established.

viii) *S* is executed as specified in Subclause 10.3, "<routine invocation>".

11) When a <routine> is called by an SQL-agent, let PD_i be the <parameter declaration> of the *i*-th parameter and let DT_i and PN_i be the <predefined type> and the <parameter name> specified in PD_i, respectively. Let PI_i be the *i*-th parameter in the procedure call.

Case:

a) If the <caller language clause> of the <routine> specifies ADA, then:

i) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i in a <general value specification> has the value PI_i.

ii) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value SV_i to PI_i.

b) If the <caller language clause> of the <routine> specifies C, then:

i) If DT_i specifies BIT(*L*), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as:

SUBSTRING (CAST (P I_i AS BIT VARYING(*ML*)) FROM 1 FOR *L*)

where *ML* is the implementation-defined maximum length of a BIT VARYING data type.

2) Let BL_i be the length in bits of PI_i. Let *BL* be the implementation-defined number of bits in a C character. Let *OL* be the smallest integer not less than the quotient of BL_i/*BL*. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value

CAST (SV_i AS CHARACTER(*OL*))

to PI_i.

ii) If DT_i specifies CHARACTER(*L*) or CHARACTER VARYING(*L*), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as an SQL character type value in the specified character set in which the octets of PI_i are the corresponding octets of that value.

When such a reference is evaluated:

A) If DT_i specifies CHARACTER(*L*) and some C character preceding the least significant C character of the value PI_i contains the implementation-defined null character that terminates a C character string, then the remaining characters of the value are set to <space>s.

B) If DT_i specifies CHARACTER(*L*) VARYING, then the length in characters of the value is set to the number of characters of PI_i that precede the implementation-defined null character that terminates a C character string.

C) If the least significant C character of the value PI_i does not contain the implementation-defined null character that terminates a C character string, then an exception condition is raised: *data exception—unterminated C string*. Otherwise, that least significant C character does not correspond to any character in PI_i and is ignored.

2) Let CL_i be one greater than the maximum possible length in octets of PN_i. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns a value that is an SQL CHARACTER(CL_i) data type in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length CL_i, concatenated with a single implementation-defined null character that terminates a C character string.

iii) If DT_i specifies BOOLEAN then . . .

****Editor's Note****

Paper SLC-55a left the preceding rule ''dangling'' with the remark: Note to editor: this needs sorting out—but is SQL3 stuff

iv) Otherwise,

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i has the value $P\ \ I_i$.

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value SV_i to PI_i.

c) If the <caller language clause> of the <routine> specifies COBOL, then

Case:

i) If DT_i specifies BIT(*L*), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as:

SUBSTRING (CAST (PI_i AS BIT VARYING(*ML*)) FROM 1 FOR *L*)

where *ML* is the implementation-defined maximum length of a BIT VARYING data type.

2) Let BL_i be the length in bits of PI_i. Let *BL* be the implementation-defined number of bits in a COBOL character. Let *OL* be the smallest integer not less than the quotient of BL_i/*BL*. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value

CAST (SV_i AS CHARACTER(*OL*))

to PI_i.

ii) If DT_i specifies CHARACTER(*L*), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as an SQL character type value in the specified character set in which the octets of PI_i are the corresponding octets of that value.

2) Let CL_i be the maximum possible length in octets of PN_i. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns a value that is an SQL CHARACTER(CL_i) data type in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length CL_i.

iii) If DT_i specifies BOOLEAN then . . .

****Editor's Note****

Paper SLC-55a left the preceding rule ''dangling'' with the remark: Note to editor: this needs sorting out—but is SQL3 stuff

iv) Otherwise,

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i has the value PI_i.

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value SV_i to PI_i.

d) If the <caller language clause> of the <routine> specifies FORTRAN, then

Case:

i) If DT_i specifies BIT(*L*), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as:

SUBSTRING (CAST (P I_i AS BIT VARYING(*ML*)) FROM 1 FOR *L*)

where *ML* is the implementation-defined maximum length of a BIT VARYING data type.

2) Let BL_i be the length in bits of PI_i. Let *BL* be the implementation-defined number of bits in a Fortran character. Let *OL* be the smallest integer not less than the quotient of BL_i/*BL*. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value

CAST (SV_i AS CHARACTER(*OL*))

to PI_i.

ii) If DT_i specifies CHARACTER(*L*), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated an SQL character type value in the specified character set in which the octets of PI_i are the corresponding octets of that value.

2) Let CL_i be the maximum possible length in octets of PN_i. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns a value that is an SQL CHARACTER(CL_i) data type in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length CL_i.

iii) If DT_i specifies BOOLEAN then . . .

****Editor's Note****

Paper SLC-55a left the preceding rule ''dangling'' with the remark: Note to editor: this needs sorting out—but is SQL3 stuff

iv) Otherwise,

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i has the value PI_i.

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value SV_i to PI_i.

e) If the <caller language clause> of the <routine> specifies MUMPS, then

Case:

i) If DT_i specifies CHARACTER VARYING(*L*), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as an SQL character type value in the specified character set in which the octets of PI_i are the corresponding octets of that value.

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns a value that is an SQL CHARACTER VARYING(*ML*) data type in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length CL_i. *ML* is the implementation-defined maximum length of variable-length character strings.

ii) Otherwise, DT_i specifies INT, DEC, or REAL, and:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as

CAST (PI_i AS DT_i)

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value

CAST (SV_i AS CHARACTER VARYING(*ML*))

to PI_i, where *ML* is the implementation-defined maximum length of variable-length character strings.

f) If the <caller language clause> of the <routine> specifies PASCAL, then

Case:

i) If DT_i specifies BIT(*L*), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as:

SUBSTRING (CAST (PI_i AS BIT VARYING(*ML*)) FROM 1 FOR *L*)

where *ML* is the implementation-defined maximum length of a BIT VARYING data type.

2) Let BL_i be the length in bits of PI_i. Let *BL* be the implementation-defined number of bits in a Pascal character. Let *OL* be the smallest integer not less than the quotient of BL_i/*BL*. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value

CAST (SV_i AS CHARACTER (*OL*))

to PI_i.

ii) If DT_i specifies CHARACTER(*L*), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated an SQL character type value in the specified character set in which the octets of PI_i are the corresponding octets of the value.

2) Let CL_i be the maximum possible length in octets of PN_i. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns a value which is an SQL CHARACTER(CL_i) data type in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length CL_i.

iii) If DT_i specifies BOOLEAN then . . .

****Editor's Note****
Paper SLC-55a left the preceding rule ''dangling'' with the remark: Note to editor: this needs sorting out—but is SQL3 stuff

iv) Otherwise,

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i has the value $P \quad I_i$.

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value SV_i to PI_i.

g) If the <caller language clause> of the <routine> specifies PLI, then

Case:

i) If DT_i specifies CHARACTER(L) or CHARACTER VARYING(L), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated an SQL character type value in the specified character set in which the octets of PI_i are the corresponding octets of the value.

2) Let CL_i be the maximum possible length in octets of PN_i. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns a value that is:

A) if DT_i specified CHARACTER(L), then an SQL CHARACTER(CL_i) data type

B) otherwise, an SQL CHARACTER VARYING(CL_i) data type

in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length CL_i.

ii) If DT_i specifies BOOLEAN then . . .

****Editor's Note****
Paper SLC-55a left the preceding rule ''dangling'' with the remark: Note to editor: this needs sorting out—but is SQL3 stuff

iii) Otherwise,

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i has the value PI_i.

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value SV_i to PI_i.

12) If ASYNC is not specified, or if *S* is a <test completion statement>, a <commit statement>, or a <rollback statement>, or if the SQL-implementation has decided to execute *S* synchronously, then wait until the execution of *S* has completed.

13) If the execution of *S* resulted in an active exception condition, then any status parameters are set to the values specified for the condition in Clause 24, "Status codes".

14) If there is more than one status parameter, then the order in which values are assigned to these status parameters is implementation-dependent.

15) If the non-dynamic or dynamic execution of an <SQL data statement> or the execution of an <SQL dynamic data statement>, <dynamic select statement>, or <dynamic single row select statement> occurs within the same SQL-transaction as the non-dynamic or dynamic execution of an SQL-schema statement and this is not allowed by the SQL-implementation, then an exception condition is raised: *invalid transaction state*.

16) If the cursor mode of the current SQL-transaction is cascade off and the <SQL procedure statement> is other than an <open statement>, a <fetch statement>, a <close statement>, a <select statement: single row>, an <SQL data change statement>, a <dynamic open statement>, a <dynamic fetch statement>, a <dynamic close statement>, a <dynamic delete statement: positioned>, or a <dynamic update statement: positioned>, then an exception condition is raised: *invalid SQL statement*.

• 1 Rule deleted.

17) When a <routine> is called by an SQL-agent, let PD_i be the <parameter declaration> of the *i*-th parameter and let DT_i and PN_i be the <data type> and the <parameter name> specified in PD_i, respectively. Let PI_i be the *i*-th parameter in the procedure call.

18) If *S* is a <select statement: single row> or a <fetch statement> and a completion condition *no data* is raised or an exception condition is raised, then the value of each PI_i for which PN_i is referenced in a <target specification> in *S* is implementation-dependent.

• 1 sentence deleted.

19) Case:

a) If *S* executed successfully, then

i) If there is more than one status parameter, then the order in which values are assigned to these status parameters is implementation-dependent.

ii) Either a completion condition is raised: *successful completion*, or a completion condition is raised: *warning*, or a completion condition is raised: *no data*, as determined by the General Rules in this and other Subclauses of this
ANSI American
ISO International
Standard.

b) If *S* did not execute successfully, then:

i) All changes made to SQL-data or schemas by the execution of *S* are canceled.

ii) The status parameter(s) is (are) set to the value(s) specified for the condition in clause Clause 24, "Status codes". If there are more two status parameters, then the order in which values are assigned to these status parameters is implementation-dependent.

20) Case:

a) If *S* is an <SQL diagnostics statement>, then the diagnostics area is not updated.

b) If *S* was executed synchronously, then diagnostics information resulting from the execution of *S* is placed into the diagnostics area as specified in Clause 20, "Diagnostics management".

c) Otherwise, no further action is taken at this time, but whenever a subsequent <test completion statement> has successfully tested the completion of *S*, then the diagnostics area is updated as required by Subclause 24.1, "SQLSTATE".

14 Control statements

14.1 <call statement>

Function

Invoke a procedure.

Format

```
<call statement> ::=
       CALL <routine invocation>
```

Syntax Rules

1) A <call statement> shall contain a <routine invocation> whose candidate <routine>s are procedures.

   Note: The *candidate <routine>s* of a <routine invocation> are defined in Subclause 10.3, "<routine invocation>".

Access Rules

1) The applicable privileges shall include the EXECUTE privilege on the <routine> identified by <routine name>.

General Rules

1) The <routine invocation> is executed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <call statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.2 <return statement>

Function

Return a value from an SQL function.

Format

```
<return statement> ::=
       RETURN <return value>

<return value> ::=
         <tuple constructor>
       | <value expression>

<tuple constructor> ::=
       TUPLE <left paren> <value expression list> <right paren>

<value expression list> ::=
       <value expression> [ { <comma> <value expression> }... ]
```

Syntax Rules

1) A <return statement> shall be contained in a <routine> that is an SQL function. Let *F* be that <routine>.

2) Case:

   a) If the <returns type> of *F* specifies a <value expression>, then the data type of the <value expression> shall be the same as the <data type> specified with RETURNS in *F*.

   b) If the <returns type> of *F* specifies a <tuple constructor>, then:

      i) *F* shall be contained in an <abstract data type definition> *ADT*. Let *TD* be the sequence of stored columns in *ADT*.

      ii) The <data type> specified in the <returns clause> of *F* shall identify the <abstract data type name> of *ADT*.

      iii) The number of <value expression>s simply contained in the <tuple constructor> shall be the same as the number of <component definition>s contained in *TD*.

      iv) Let TU_i be the *i*-th <value expression> in the <tuple constructor> and let TD_i be the data type of the *i*-th column of *TD*. CAST (TU_i AS TD_i) shall conform to the Syntax Rules of Subclause 6.13, "<cast specification>".

Access Rules

None.

General Rules

1) Case:

   a) If the <returns type> is a <value expression>, then the return value is set to the value of that <value expression>.

   b) If the <returns type> is a <tuple constructor>, then:

      i) For every <value expression> in the <tuple constructor>, the General Rules for CAST (TU_i AS TD_i) are performed.

      ii) The return value is set to a value whose type is that of *TN* and whose *i*-th component value is CAST (TU_i AS TD_i).

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <return statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.3 <compound statement>

Function

Specify a statement that groups other statements together.

Format

```
<compound statement> ::=
       [ <beginning label> <colon> ]
       [ DECLARE <local declaration list> ]
       BEGIN [ [ NOT ] ATOMIC ]
       [ <SQL statement list> ]
       [ <exception handler> ]
       END [ <ending label> ]

<beginning label> ::= <statement label>

<ending label> ::= <statement label>

<statement label> ::= <identifier>

<SQL statement list> ::= <terminated SQL statement>...
```

ANSI Only--SQL3

```
<local declaration list> ::= <local declaration>...

<local declaration> ::=
         <SQL variable declaration>
       | <routine>
       | <declare cursor>
       | <temporary table declaration>
       | <temporary view definition>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
<local declaration list> ::= <SQL variable declaration>...
```

```
<terminated SQL statement> ::=
       <SQL procedure statement> <semicolon>
```

Syntax Rules

1) The <SQL procedure statement> shall not be a <commit statement>, a <rollback statement>, or an <SQL connection statement>.

2) If a <compound statement> has a <beginning label>, it is called a labeled <compound statement>.

3) If an <ending label> is specified, then an identical <beginning label> shall be specified.

4) If a <beginning label> is specified, then it shall be different from all other <statement label>s immediately contained in a
ANSI <compound statement>, a <loop statement>, or a <for statement>
ISO <compound statement> or a <loop statement>
in the containing <routine>.

5) If neither ATOMIC nor NOT ATOMIC is specified, then ATOMIC is implicit.

6) Case:

   a) If the <compound statement> is contained in an <exception handler>, then it shall not contain an <exception handler>.

   b) Otherwise, if <exception handler> is not specified, then the following <exception handler> is implicit:

      EXCEPTION WHEN OTHERS THEN RESIGNAL ;

7) If an explicit <beginning label> is not specified and the <compound statement> is the outermost <statement> of a <routine> then the <procedure name> of that <routine> is the implicit <beginning label>.

ANSI Only–SQL3

8) The ''declared local name'' of an <SQL variable declaration> *VD*, a <routine> *R*, a <declare cursor> *DC*, a <temporary table declaration> *TTD*, or a <temporary view declaration> *TVD*, in a <local declaration list> is the <SQL variable name>, <routine name>, <cursor name>, or <table name> immediately contained in *VD*, *R*, *DC*, *TTD*, or *TVD*.

9) If two <local declaration>s in a <local declaration list> have the same declared local name, then both of those <local declaration>s shall be <routine>s.

10) If a <compound statement> *CS* is simply contained in a <routine>, then no <local declaration> in the <local declaration list> of *CS* shall have a declared local name that is equal to the <parameter name> of any <parameter declaration> in the <parameter declaration list> of the <routine>.

Access Rules

None.

General Rules

ANSI Only–SQL3

1) The variables, cursors, routines, and temporary tables and view specified in the <local declaration list> are created in an implementation-dependent order.

ISO Only—caused by ANSI changes not yet considered by ISO

2) The variables specified in <local declaration list> are created in any order. If an <SQL variable declaration> specifies <default clause>, then the value of that variable is set to the default value according to the General Rules of Subclause 11.5, "<default clause>"; otherwise, the value of the variable is implementation-dependent.

3) The <SQL procedure statement>s of the <SQL statement list> are executed in the order in which they are specified.

4) If there is an active exception condition *E* at the completion of any of these <SQL procedure statement>s, then:

   a) Execution of all of the <SQL procedure statement>s is terminated immediately. If an <exception handler> has been specified, then control is passed to that <exception handler>.

   b) If ATOMIC has been specified and if the <exception handler> completes with an active exception condition, then all changes made to SQL-data or schemas by the execution of every SQL-statement contained in the <SQL statement list> and any <SQL procedure statement>s triggered by the execution of any such statement are cancelled.

5) The
   ANSI variables, cursors, routines, temporary tables, and temporary views
   ISO variables
   specified in <local declaration list> are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <compound statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.4 <exception handler>

Function

Provide exception handlers for a block statement.

Format

```
<exception handler> ::=
       EXCEPTION [ <exception handler contents>... ]

<exception handler contents> ::=
       WHEN <exception handler condition>
         THEN <exception handler action>

<exception handler condition> ::=
         <exception name list>
       | OTHERS

<exception handler action> ::=
       <SQL statement list>

<exception name list> ::=
       <exception name> [ { <comma> <exception name> }... ]
```

Syntax Rules

1) The <SQL statement list> in an <exception handler action> shall not contain a <leave statement> that specifies a <statement label> corresponding to a <statement> outside the containing <compound statement>.

2) OTHERS shall not be specified more than once in the <exception handler>.

3) If <exception handler> is specified that does not contain an <exception handler contents> that specifies OTHERS, then the following <exception handler contents> is implicitly added to the <exception handler>:

   WHEN OTHERS THEN RESIGNAL ;

4) The <exception handler> shall not contain more than one occurrence of any <exception name> that has the same <exception value>.

Access Rules

None.

General Rules

1) Let *E* be the active exception condition that caused execution to proceed into the <exception handler>. If an <exception handler contents> *EHC* specifies an <exception handler condition> that contains an <exception name> in its <exception name list> that has associated with it a value that is equal to the value specified by *E*, then *EHC* will be executed. Otherwise, the <exception handler contents> that specifies OTHERS will be executed.

2) The execution of an <exception handler contents> causes the <SQL procedure statement>s of its <SQL statement list> to be executed sequentially.

3) Case:

   a) If a <leave statement> is executed with a <statement label> that corresponds to the containing <compound statement>, then the <compound statement> terminates with no active condition.

   b) If any <SQL procedure statement> contained in the <SQL statement list> completes execution with an active exception statement *E1*, then the execution of the <SQL statement list> terminates. Execution continues as though the <exception handler> terminated with the <resignal statement>:

```
RESIGNAL E1;
```

   c) Otherwise, when the <exception handler action> completes execution, the <compound statement> terminates with no active condition.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <exception handler>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.5 <exception declaration>

Function

Declare an exception name and a corresponding SQLSTATE value.

Format

```
<exception declaration> ::=
       DECLARE <exception name> EXCEPTION [ FOR ]
         <exception value> <semicolon>

<exception value> ::=
       SQLSTATE [ VALUE ] <character string literal>
```

Syntax Rules

1) The <exception name> shall be different from the <exception name> in any other <exception declaration> in the containing <module>.

2) The <character string literal> provided in an <exception value> shall be an SQLSTATE value that conforms to the rules of Subclause 24.1, "SQLSTATE".

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <exception declaration>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.6 <SQL variable declaration>

Function

Declare a variable.

Format

```
<SQL variable declaration> ::=
       <SQL variable name list> [ <constant or updatable> ]
         { <data type> | <domain name> } [ <default clause> ] <semicolon>

<SQL variable name list> ::=
       <SQL variable name> [ { <comma> <SQL variable name> }... ]
```

Syntax Rules

1) If neither CONSTANT nor UPDATABLE is specified, then UPDATABLE is implicit.

2) For each <SQL variable name> in the <SQL variable name list>:

   a) If CONSTANT is specified, then a <default clause> shall be specified.

   b) The effective data type of the variable declared with a <domain name> is the base data type of the domain and is not subject to any constraints that may apply to the domain.

ISO Only—caused by ANSI changes not yet considered by ISO

   c) The <SQL variable name> shall be different from the <SQL variable name> in any other <SQL variable declaration> in the containing <local declaration list>.

   d) If an <SQL variable declaration> is simply contained in a <compound statement> that is the outermost <compound statement> of a <routine>, then the <SQL variable name> shall be different from the <parameter name> of any <parameter declaration> in the <routine>.

Access Rules

None.

General Rules

ANSI Only–SQL3

1) When the variable associated with the <SQL variable declaration> is created, if the <SQL variable declaration> specifies <default clause>, then the value of that variable is set to the default

value according to the General Rules of Subclause 11.5, "<default clause>"; otherwise, the value of the variable is implementation-dependent.

ISO Only—caused by ANSI changes not yet considered by ISO

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <SQL variable declaration>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.7 <assignment statement>

Function

Assign a value to a component of an abstract data type or to a local variable.

Format

```
<assignment statement> ::=
       SET <assignment target> <equals operator> <value expression>

<assignment target> ::=
         <attribute reference>
       | <target specification>
```

Syntax Rules

1) If the <assignment target> is an <attribute reference>, then the contained <column name> shall not be OID.

2) The <assignment target> shall not be read-only.

3) If the <assignment target> is constant, then it shall identify a stored attribute and the <assignment statement> shall be contained in a <routine> that is a constructor for the abstract data type instance containing the component identified by <assignment target>.

4) If the <assignment target> is an <attribute reference> that identifies a stored attribute or is a <SQL variable name>, then the data type of the <value expression> shall be assignable to the data type of the abstract data type component identified by <assignment target>.

5) The <value specification> contained in the <attribute reference> shall be a <target specification> or <simple target specification>.

Access Rules

None.

General Rules

1) Case:

   a) If the <assignment target> *AT* identifies a virtual attribute, then let *I* be the instance, let *SF* be the set function identified by *AT*, and let *V* be the value of the <value expression>; the value of the virtual column is determined by the SQL function invocation *SF*(*I*,*V*).

ANSI Only–SQL3

---

   b) Otherwise, let *TV* be the item identified by the <assignment target>, and *SV* be the value of the <value expression>. The General Rules of Subclause 9.1, "Retrieval assignment", are applied to *TV* and *SV* as *TARGET* and *VALUE* respectively.

ISO Only—caused by ANSI changes not yet considered by ISO

c) Otherwise, the item identified by <assignment target> is set to the value of the <value expression>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <assignment statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.8 <new statement>

Function

Create a new abstract data type instance.

Format

```
<new statement> ::=
```

ANSI Only--SQL3

```
      NEW <item reference>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
      NEW <new object name>

<new object name> ::= <local variable name>
```

Syntax Rules

1) <new statement> shall be specified only in a <routine> that specifies CONSTRUCTOR.

ANSI Only–SQL3

2) The <item reference> shall be an SQL variable reference that identifies a local variable declared in the containing <abstract data type definition> or <table definition>.

Note: ''SQL variable reference'' is defined in Subclause 6.3, "<item reference>".

ISO Only—caused by ANSI changes not yet considered by ISO

3) <new object name> shall identify a local variable declared in the containing <routine> with a data type of object identifier type that REFERENCES the containing abstract data type or table.

Access Rules

None.

General Rules

1) A new ADT instance whose data type is the data type of the containing <abstract data type definition> or <table definition> is created and is assigned to the local variable *LV* identified by the <new object name>.

2) If *LV* has a column named OID, then assign to it a unique object identifier value.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <new statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.9 <destroy statement>

Function

Destroy an abstract data type instance.

Format

```
<destroy statement> ::=
       DESTROY <object parameter name>

<object parameter name> ::= <parameter name>
```

Syntax Rules

1) <destroy statement> shall be specified only in a <routine> that specifies DESTRUCTOR.

2) <object parameter name> shall be the <parameter name> of a parameter in the <parameter list> of the containing <routine> and shall have a data type that is the same as the data type of the containing <abstract data type definition> or <table definition>.

Access Rules

None.

General Rules

1) If the <object parameter name> does not identify a valid abstract data type instance, then an exception condition is raised: *invalid ADT instance*.

2) Let the value of the <object parameter name> be *OBJ*, and let *T* be the <table name> of the table specified in the <delete statement: searched> from which the <SQL function> containing this <destroy statement> was invoked. The object identified by *OBJ* is marked for deletion in *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <destroy statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.10 <case statement>

Function

Select an execution path based on multiple cases.

Format

```
<case statement> ::=
       CASE <value expression>
         <case statement when clause>...
         [ <case statement else clause> ]
       END CASE

<case statement when clause> ::=
       WHEN { <literal> | NULL } THEN <SQL statement list>

<case statement else clause> ::=
       ELSE <SQL statement list>
```

Syntax Rules

1) The data type of the <value expression> and the data type of each <literal> immediately contained in a <case statement when clause> shall be comparable. <value expression> shall not directly contain a <column specification>.

2) No <literal> immediately contained in a <case statement when clause> shall have a value equal to any other <literal> immediately contained in a <case statement when clause> that is contained in the same <case statement>.

3) NULL shall be specified for no more than one <case statement when clause> of a <case statement>.

Access Rules

None.

General Rules

1) Let *CS* be the <case statement>.

2) Let *VE* be the <value expression> specified for *CS*.

3) Case:

   a) If the value of *VE* is null and if NULL is specified in any <case statement when clause> in *CS*, then the <SQL statement list> specified for the <case statement when clause> in *CS* is executed.

   b) If the value of *VE* is not null and if the value of a <literal> specified in any <case statement when clause> in *CS* is the same as the value of *VE*, then the <SQL statement list> specified for the <case statement when clause> in *CS* is executed.

   c) Otherwise:

Case:

i) If a <case statement else clause> is specified for *CS*, then the <SQL statement list> specified for the <case statement else clause> is executed.

ii) If a <case statement else clause> is not specified for *CS*, then an exception condition is raised: *case not found for case statement*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <case statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.11 <if statement>

Function

Provide conditional execution based on the truth value of a condition.

Format

```
<if statement> ::=
       IF <search condition>
         <if statement then clause>
         [ <if statement elseif clause>... ]
         [ <if statement else clause> ]
       END IF

<if statement then clause> ::=
       THEN <SQL statement list>

<if statement elseif clause> ::=
       ELSEIF <search condition> THEN <SQL statement list>

<if statement else clause> ::=
       ELSE <SQL statement list>
```

Syntax Rules

1) If one or more <if statement elseif clause>s are specified, then the <if statement> is equivalent to an <if statement> that does not contain ELSEIF by performing the following transformation recursively:

```
IF <search condition>
  <if statement then clause>
  <if statement elseif clause 1>
  [ <if statement elseif clause>  . . .  ]
  [ <if statement else clause> ]
END IF
```

is equivalent to

```
IF <search condition>
  <if statement then clause>
  ELSE
    IF <search condition 1>
      THEN <statement list 1>
      [ <if statement elseif clause>... ]
      [ <if statement else clause> ]
    END IF
END IF
```

where <search condition 1> is the <search condition> directly contained in <if statement elseif clause 1> and <statement list 1> is the <SQL statement list> directly contained in <if statement elseif clause 1>.

Access Rules

None.

General Rules

1) If the <search condition> directly contained in the <if statement> evaluates to true, then the <SQL statement list> directly contained in the <if statement then clause> is executed and execution of the <if statement> is terminated.

2) If a <if statement else clause> is specified, then the <SQL statement list> directly contained in the <if statement else clause> is executed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <if statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.12 <leave statement>

Function

Continue execution by leaving a block or loop statement.

Format

```
<leave statement> ::=
       LEAVE <statement label>
```

Syntax Rules

1) A <leave statement> *L* shall be contained within a labeled <SQL procedure statement> *S*.

2) The <statement label> from *L* must be identical to the <label> from *S*. This <statement label> is called the *corresponding <statement label>*.

Access Rules

None.

General Rules

1) Let *L* be the <leave statement> that is encountered.

2) Let *S* be the <SQL procedure statement> containing *L* with a corresponding <statement label>.

3) No further action is performed by *S*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <leave statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.13 <loop statement>

Function

Repeat the execution of a statement.

Format

```
<loop statement> ::=
       [ <beginning label> <colon> ]
       [ <loop iteration scheme> ]
       LOOP
         <SQL statement list>
       END LOOP [ <ending label> ]

<loop iteration scheme> ::=
       WHILE <search condition>
```

Syntax Rules

1) If a <loop statement> has a <beginning label>, then it is called a *labeled <loop statement>*.

2) If <ending label> is specified, then it shall be identical to <beginning label>.

Access Rules

None.

General Rules

1) Let *LS* be the <loop statement>.

2) If a <loop iteration scheme> *I* is provided for *LS*, then:

   a) The <search condition> contained in *I* is evaluated.

   b) If the <search condition> is true, then the <SQL statement list> is executed, and if each statement in the <SQL statement list> executes without raising an exception, then execution of *LS* is repeated.

   c) If the <search condition> is false or unknown, then the execution of *LS* is terminated.

3) If a <loop iteration scheme> is not provided for *LS*, then the <SQL statement list> is executed repeatedly until an exception condition is raised or execution of a <leave statement> terminates execution of *LS*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <loop statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

ANSI Only–SQL3

14.14 <for statement

Function

Execute a statement for each row of a query expression.

Format

```
<for statement> ::=
      [ <beginning label> <colon> ]
        FOR <for loop name> IN <for loop specification>
          DO <SQL statement list>
      END FOR [ <ending label> ]

<for loop name> ::= <identifier>

<for loop specification> ::=
      <query expression> [ <order by clause> ]
```

Syntax Rules

1) Let *FLS* be the <for loop specification> of a <for statement>, *FS*. Let *CN* be an implementation-dependent <cursor name> that is different from any <cursor name> in the outermost containing <module> or <routine>. Let *DC* be the following <declare cursor>:

   DECLARE *CN* INSENSITIVE CURSOR FOR *FLS* FOR READ ONLY

2) Let *QE* be the <query expression> of *FLS*. Each column of the table specified by *QE* shall have a <column name>. Let *V1*, *V2*, . . . , *VN* be those <column name>s. Let *DT1*, *DT2*, . . . , *DTN* be the data types of the respective columns.

3) Let *BL*, *FLN*, and *SLL* be the <beginning label>, <for loop name>, and <SQL statement list> of *FS*. Let *S* be an implementation-dependent <variable name> that is different from any <variable name> in the outermost containing <module> or <routine>. The <for statement> is equivalent to:

```
      FLN:
          DC;
          DECLARE :S CHARACTER (5);
          DECLARE :V1 DT1;
          DECLARE :V2 DT1;
      .
      .
      .
          DECLARE :VN DTN;

  BEGIN
          OPEN CN;
          FETCH CN INTO :V1, :V2 . . . , :VN;
          GET DIAGNOSTICS EXCEPTION 1 :S = RETURNED_SQLSTATE;
```

```
    BL:
      WHILE :S <> '02000' LOOP
      SLL;
      FETCH CN INTO :V1, :V2 . . . , :VN;
      GET DIAGNOSTICS EXCEPTION 1 :S = RETURNED_SQLSTATE;
    END LOOP BL;
END FLN;
```

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL Language shall contain no <for statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.15 <signal statement>

Function

Signal an exception condition.

Format

```
<signal statement> ::=
       SIGNAL <exception name>
```

Syntax Rules

1) <exception name> shall be defined in the scope of this statement by an <exception declaration>.

Access Rules

None.

General Rules

1) Let *N* be the value of the statement information field NUMBER in the Diagnostics Area before the execution of the <signal statement>. The existing exception information areas 1 through *N* in the Diagnostics Area are cleared. The value of the statement information field NUMBER in the Diagnostics Area is set to 1 and the MORE field is set to 'N'.

   The statement information field COMMAND_FUNCTION is set to ''<signal statement>'' and the DYNAMIC_FUNCTION field is cleared. In the first exception information area in the Diagnostics Area, the exception information field RETURNED_SQLSTATE is set to the value associated with the specified <exception name> and the field EXCEPTION_IDENTIFIER is set to contain the <exception name>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <signal statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.16 <resignal statement>

Function

Resignal an exception condition.

Format

```
<resignal statement> ::=
       RESIGNAL [ <exception name> ]
```

Syntax Rules

1) A <resignal statement> shall be contained in an <exception handler action>.

2) If <exception name> is specified, then it shall be defined in the scope of this statement by an <exception declaration>.

Access Rules

None.

General Rules

1) Let *N* be the value of the statement information field NUMBER in the Diagnostics Area before the execution of the <resignal statement>.

   Case:

   a) If <exception name> is not specified, then the Diagnostics Area remains unchanged and the exception information 1 through *N* is propagated (resignaled).

   b) If <exception name> is specified, then the statement information NUMBER field is incremented. All existing exception information is stacked such that the *i*-th exception information area is placed at the position of the *i*+1-st exception information area in the Diagnostics Area. If the maximum number of exception information areas for the Diagnostics Area is exceeded, the statement information fields NUMBER and MORE are modified as specified in Subclause 14.4, "<exception handler>", and the last exception information area is lost.

      In the first exception information area in the Diagnostics Area, the exception information field RETURNED_SQLSTATE is set to the value associated with the specified <exception name> and the field EXCEPTION_IDENTIFIER is set to contain the <exception name>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <resignal statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

15 Data manipulation

15.1 <declare cursor>

Function

Define a cursor.

Format

```
<declare cursor> ::=
       DECLARE <cursor name> [ { SENSITIVE | INSENSITIVE } ] [ SCROLL ] CURSOR
         FOR <cursor specification>

<cursor specification> ::=
       <query expression> [ <order by clause> ]
         [ <updatability clause> ]

<updatability clause> ::=
       FOR { READ ONLY | UPDATE [ OF <column name list> ] }

<order by clause> ::=
       ORDER BY <sort specification list>

<sort specification list> ::=
       <sort specification> [ { <comma> <sort specification> }... ]

<sort specification> ::=
       <sort key> [ <collate clause> ] [ <ordering specification> ]

<sort key> ::=
       <value expression>

<ordering specification> ::= ASC | DESC
```

Syntax Rules

ANSI Only–SQL3

1) If a <declare cursor> is contained in a <module> *M* and is not contained in a <compound statement>, then:

   a) The <cursor name> shall be different from the <cursor name> of any other <declare cursor> in *M*.

b) Any <parameter name> contained in the <cursor specification> shall be defined in a <parameter declaration> in the <routine> in *M* that contains an <open statement> that specifies the <cursor name>.

ISO Only—caused by ANSI changes not yet considered by ISO

2) The <cursor name> shall be different from the <cursor name> contained in any other <declare cursor> in the same <module>.

3) Any <parameter name> contained in the <cursor specification> shall be defined in a <parameter declaration> in the <routine> in the containing <module> that contains an <open statement> that specifies the <cursor name>.

Note: See the Syntax Rules of Subclause 13.1, "<module>".

4) When <cursor name> is referenced in an <update statement: positioned>, no <object column> in the <set clause> shall identify a column that is specified in a <sort specification> of an <order by clause>.

5) Let *T* be the table specified by the <query expression>.

6) Let *CS* be the cursor specified by the <cursor specification>.

7) If <updatability clause> is not specified, then:

   a) If either INSENSITIVE, SCROLL, or ORDER BY is specified, or if *T* is a read-only table, then an <updatability clause> of READ ONLY is implicit.

   b) Otherwise, an <updatability clause> of FOR UPDATE without a <column name list> is implicit.

8) If an <updatability clause> of READ ONLY is specified or implicit, then *CS* is a *read-only cursor*; otherwise, *CS* is an *updatable cursor*.

9) If an <order by clause> is specified, then the cursor specified by the <cursor specification> is said to be an *ordered cursor*.

10) The *simply underlying table* of a <cursor specification> is the table derived from by the <query expression> simply contained in the <cursor specification>.

11) If *T* is an updatable table, then let *TU* be the leaf generally underlying table of the <cursor specification>.

12) If an <order by clause> is specified, then let K_i be the <sort key> contained in the *i*-th <sort specification>. Each row of *ST* contains the values of the columns of *T* and the values of the <sort key>s for the corresponding row of *T*, computed as follows:

    Case:

    a) If K_i is an <unsigned integer>, then the value *N* of the <unsigned integer> shall be greater than 0 and less than or equal to the degree of *T*. Let *C* be the column of *ST* whose ordinal position is *N*. The sort key value V_i is the value of *C* in a row of *ST*.

b) Otherwise, K_i is a <value expression>. The <value expression> shall not contain a <subquery> or a <set function specification> but shall contain a <column reference>.

i) Let *X* be any <column reference> directly contained in K_i.

ii) Let *QE* be the <query expression> immediately contained in the <cursor specification>. If *QE* is a <query primary> that is a <simple table> that is a <query specification>, then the <cursor specification> is said to be a *simple table query*. If the <cursor specification> is a simple table query, then let *TE* be the <table expression> contained in the <query specification> contained in *QE*.

iii) If *X* contains a <qualifier>, then the <cursor specification> shall be a simple table query and *X* shall unambiguously identify a column of *TE*. The value of that column is used to compute the value of K_i.

iv) If *X* does not contain a <qualifier>, then

Case:

1) If the <cursor specification> is not a simple table query, then *X* shall be identical to the name of exactly one column of *T*. The value of that column is used to compute the value of K_i.

2) Otherwise, the <cursor specification> is a simple table query.

A) *X* shall not be identical to the name of more than one column of *T*.

B) If *X* is identical to the name of a column of *T*, then the value of that column is used to compute the value of K_i.

C) If *X* is not identical to the name of any column of *T*, then *X* shall be identical to the name of exactly one column of *TE*. The value of that column is used to compute the value of K_i.

v) If the <cursor specification> is a simple table query and if the <query specification> contained in *QE* specifies the <set quantifier> DISTINCT or directly contains one or more <set function specifiction>s or if *T* is a grouped table, then there shall be a <derived column> equivalent to *X* in the <select list> of the <query specification>.

vi) The sort key value V_i is the value of K_i for a row of *T*.

13) If a <sort specification> contains a <collate clause>, then the data type of the column identified by the <sort specification> shall be character string. The column descriptor of the corresponding column in the result has the collating sequence specified in <collate clause> and the coercibility attribute *Explicit*.

14) If an <updatability clause> of FOR UPDATE with or without a <column name list> is specified, then *T* shall be an updatable table and INSENSITIVE shall not be specified.

15) If an <updatability clause> of FOR UPDATE without a <column name list> is specified or implicit, then a <column name list> that includes the <column name> of every column of *TU* is implicit.

16) If an <updatability clause> of FOR UPDATE with a <column name list> is specified, then each <column name> in the <column name list> shall be the <column name> of a column of *TU*.

17) If a <sort key> simply contains a <value expression> that simply contains a <column reference> that identifies a column whose data type is an abstract data type, then the <abstract data type definition> that defined that abstract data type shall contain a <less-than clause> that does not specify NONE.

ANSI Only–SQL3

18) The data type of a <value expression> simply contained in a <sort key> shall not be a <collection type>.

Access Rules

None.

General Rules

1) If *T* is an updatable table, then the cursor is associated with the named table *TU*. For every row in *T*, there is exactly one corresponding row in *TU* from which the row of *T* is derived.

ANSI Only–SQL3

2) If an <order by clause> is not specified, then:

Case:

a) If *T* is an ordered table, then the ordering of *T* is used by the cursor.

b) Otherwise, the ordering of rows in *T* is implementation-dependent.

ISO Only—caused by ANSI changes not yet considered by ISO

3) If an <order by clause> is not specified, then the table specified by the <cursor specification> is *T* and the ordering of rows in *T* is implementation-dependent.

4) If an <order by clause> is specified, then the ordering of rows of the result is effectively determined by the <order by clause> as follows:

a) Each <sort specification> specifies the sort direction for the corresponding sort key K_i. If DESC is not specified in the *i*-th <sort specification>, then the sort direction for K_i is ascending and the applicable <comp op> is the <less than operator>. Otherwise, the sort direction for K_i is descending and the applicable <comp op> is the <greater than operator>.

b) Let P be any row of the result table and let Q be any other row of that table, and let PV_i and QV_i be the values of K_i in these rows, respectively. The relative position of rows P and Q in the result is determined by comparing PV_i and QV_i according to the rules of Subclause 8.2, "<comparison predicate>", where the <comp op> is the applicable <comp op> for K_i, with the following special treatment of null values. Whether a sort key value that is null is considered greater or less than a non-null value is implementation-defined, but all sort key values that are null shall either be considered greater than all non-null values or be considered less than all non-null values. PV_i is said to *precede* QV_i if the value of the <comparison predicate> ''PV_i <comp op> QV_i'' is true for the applicable <comp op>.

c) In the result table, the relative position of row P is before row Q if and only if PV_n precedes QV_n for some n greater than 0 and less than the number of <sort specification>s and $PV_i = QV_i$ for all $i < n$. The relative order of two rows that are not distinct is implementation-dependent.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <declare cursor> shall not specify SENSITIVE.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <declare cursor> shall not specify INSENSITIVE.

   b) If an <updatability clause> of FOR UPDATE with or without a <column name list> is specified, then neither SCROLL nor ORDER BY shall be specified.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <declare cursor> shall not specify SCROLL.

   b) A <cursor specification> shall not contain an <updatability clause>.

15.2 <open statement>

Function

Open a cursor and set the cursor mode of the current SQL-transaction.

Format

```
<open statement> ::=
       OPEN <cursor name>
         [ <open cascade option> ]

<open cascade option> ::=
         CASCADE ON | CASCADE OFF
```

Syntax Rules

ISO Only—caused by ANSI changes not yet considered by ISO

1) The containing <module> shall contain a <declare cursor> *CR* whose <cursor name> is the same as the <cursor name> in the <open statement>.

2) If CASCADE is not specified, then CASCADE ON is implicit.

3) If CASCADE OFF is specified, then *CR* shall be updatable.

Access Rules

1) The Access Rules for the <query expression> simply contained in the <declare cursor> identified by the <cursor name> are applied.

General Rules

1) If cursor *CR* is not in the closed state, then an exception condition is raised: *invalid cursor state*.

2) If the cursor mode of the current SQL-transaction is set to *cascade off*, then an exception condition is raised: *invalid cursor mode state*.

3) Let *S* be the <cursor specification> of cursor *CR*.

4) Cursor *CR* is opened in the following steps:

   a) A copy of *S* is effectively created in which:

      i) Each <target specification> is replaced by the value of the target;

ii) Each <value specification> generally contained in *S* that is
ISO USER,
CURRENT_USER, SESSION_USER, or SYSTEM_USER is replaced by the value resulting from evaluation of CURRENT_USER, SESSION_USER, or SYSTEM_USER, respectively, with all such evaluations effectively done at the same instant in time; and

iii) Each <datetime value function> generally contained in *S* is replaced by the value resulting from evaluation of that <datetime value function>, with all such evaluations effectively done at the same instant in time.

b) Let *T* be the table specified by the copy of *S*.

c) A table descriptor for *T* is effectively created.

d) The General Rules of Subclause 15.1, "<declare cursor>", are applied.

e) Case:

i) If *S* specifies INSENSITIVE, then a copy of *T* is effectively created and cursor *CR* is placed in the open state and its position is before the first row of the copy of *T*.

ii) Otherwise, cursor *CR* is placed in the open state and its position is before the first row of *T*.

5) Let *CC* be the set of constraints that are defined by <assertion definition>s or by <table constraint definition>s of tables that are nodes in the constraint-referential-action graph for *CR*. Let *CCL* be a comma-separated list of the <constraint name>s of the constraints in *CC*.

6) Case:

a) If CASCADE ON is specified or implied, then:

i) If *CR* is a cursor that was explicitly or implicitly declared FOR UPDATE and any cursor *CR1* in the current SQL-transaction that is in the open state was opened CASCADE OFF and the constraint-referential-action graphs for *CR* and *CR1* are not disjoint, then an exception condition is raised: *invalid cursor state*.

ii) The following <set constraints mode statement> is implicitly executed:

SET *CCL* CONSTRAINTS IMMEDIATE

iii) The cursor mode of the current SQL-transaction is set to *cascade on*.

b) If CASCADE OFF is specified, then:

i) If any cursor *CR1* that is implicitly or explicitly declared FOR UPDATE in the current SQL-transaction is in the open state and the constraint-referential-action graphs for *CR* and *CR1* are not disjoint, then an exception condition is raised: *invalid cursor state*.

ii) The following <set constraints mode statement> is implicitly executed:

SET *CCL* CONSTRAINTS DEFERRED

iii) The cursor mode of the current SQL-transaction is set to *cascade off*.

****Editor's Note****

The preceding rules deal with CASCADE ON and OFF and depend on the constraint-referential-action graph, which has been deleted. Therefore, these rules will need to be revised.

7) If *CR* specifies INSENSITIVE, and the implementation is unable to guarantee that significant changes will be invisible through *CR*, then an exception condition is raised: *cursor sensitivity exception—request rejected*.

8) If *CR* specifies SENSITIVE, and the implementation is unable to guarantee that significant changes will be visible through *CR*, then an exception condition is raised: *cursor sensitivity exception—request rejected*.

9) Whether an implementation is able to disallow significant changes that would not be visible through a currently open cursor is implementation-defined.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <open statement> shall not contain an <open cascade option>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

15.3 <fetch statement>

Function

Position a cursor on a specified row of a table and retrieve values from that row.

Format

```
<fetch statement> ::=
       FETCH [ [ <fetch orientation> ] FROM ]
         <cursor name> INTO <fetch target list>

<fetch orientation> ::=
         NEXT
       | PRIOR
       | FIRST
       | LAST
       | { ABSOLUTE | RELATIVE } <simple value specification>

<fetch target list> ::=
       <target specification> [ { <comma> <target specification> }... ]
```

Syntax Rules

1) If the <fetch orientation> is omitted, then NEXT is implicit.

ANSI Only–SQL3

2) Let *CR* be the <declare cursor> denoted by the <cursor name> and let *T* be the table defined by the <cursor specification> of *CR*.

ISO Only—caused by ANSI changes not yet considered by ISO

3) The containing <module> shall contain a <declare cursor> *CR* whose <cursor name> is the same as the <cursor name> in the <fetch statement>. Let *T* be the table defined by the <cursor specification> of *CR*.

4) If the implicit or explicit <fetch orientation> is not NEXT, then
ISO the <declare cursor>
CR shall specify SCROLL.

5) If a <fetch orientation> that contains a <simple value specification> is specified, then the data type of that <simple value specification> shall be exact numeric with a scale of 0.

6) The number of <target specification>s in the <fetch target list> shall be the same as the degree of table *T*. The *i*-th <target specification> in the <fetch target list> corresponds with the *i*-th column of table *T*.

7) The Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to each corresponding <target specification> and column of table *T*, as *TARGET* and *VALUE*, respectively.

Access Rules

None.

General Rules

1) If cursor *CR* is not in the open state, then an exception condition is raised: *invalid cursor state*.

2) If the cursor mode of the current SQL-transaction is *cascade off*, and if the previous <SQL procedure statement> that specified the <cursor name> of *CR* that was executed within the current set-processing mode session of the current SQL-transaction was a <fetch statement> or a <dynamic fetch statement>, then an exception condition is raised: *invalid SQL statement*.

3) Case:

   a) If the <fetch orientation> contains a <simple value specification>, then let *J* be the value of that <simple value specification>.

   b) If the <fetch orientation> specifies NEXT or FIRST, then let *J* be +1.

   c) If the <fetch orientation> specifies PRIOR or LAST, then let *J* be −1.

4) Let T_t be a table of the same degree as *T*.

   Case:

   a) If the <fetch orientation> specifies ABSOLUTE, FIRST, or LAST, then let T_t contain all rows of *T*, preserving their order in *T*.

   b) If the <fetch orientation> specifies NEXT or specifies RELATIVE with a positive value of *J*, then:

      i) If the table *T* identified by cursor *CR* is empty or if the position of *CR* is on or after the last row of *T*, then let T_t be a table of no rows.

      ii) If the position of *CR* is on a row *R* that is other than the last row of *T*, then let T_t contain all rows of *T* ordered after row *R*, preserving their order in *T*.

      iii) If the position of *CR* is before a row *R*, then let T_t contain row *R* and all rows of *T* ordered after row *R*, preserving their order in *T*.

   c) If the <fetch orientation> specifies PRIOR or specifies RELATIVE with a negative value of *J*, then:

      i) If the table *T* identified by cursor *CR* is empty or if the position of *CR* is on or before the first row of *T*, then let T_t be a table of no rows.

      ii) If the position of *CR* is on a row *R* that is other than the first row of *T*, then let T_t contain all rows of *T* ordered before row *R*, preserving their order in *T*.

      iii) If the position of *CR* is before the next row of a row *R* that is not the last row of *T*, then let T_t contain row *R* and all rows of *T* ordered before row *R*, preserving their order in *T*.

iv) If the position of *CR* is after the last row of *T*, then let T_t contain all rows of *T*, preserving their order in *T*.

d) If RELATIVE is specified with a zero value of *J*, then:

i) If the position of *CR* is on a row of *T*, then let T_t be a table comprising that one row.

ii) Otherwise, let T_t be an empty table.

5) Let *N* be the number of rows in T_t. If *J* is positive, then let *K* be *J*. If *J* is negative, then let *K* be *N*+*J*+1. If *J* is zero and ABSOLUTE is specified, then let *K* be zero; if *J* is zero and RELATIVE is specified, then let *K* be 1.

6) Case:

a) If *K* is greater than 0 and not greater than *N*, then *CR* is positioned on the *K*-th row of T_t and the corresponding row of *T*. That row becomes the current row of *CR*.

b) Otherwise, no SQL-data values are assigned to any targets in the <fetch target list>, and a completion condition is raised: *no data*.

Case:

i) If the <fetch orientation> specifies RELATIVE with *J* equal to zero, then the position of *CR* is unchanged.

ii) If the <fetch orientation> implicitly or explicitly specifies NEXT, specifies ABSOLUTE or RELATIVE with *K* greater than *N*, or specifies LAST, then *CR* is positioned after the last row.

iii) Otherwise, the <fetch orientation> specifies PRIOR, FIRST, or ABSOLUTE or RELATIVE with *K* not greater than *N* and *CR* is positioned before the first row.

7) If a completion condition *no data* has been raised, then no further General Rules of this Subclause are applied.

8) If an exception condition is raised during derivation of any <derived column> associated with the current row of *CR*, then *CR* remains positioned on the current row.

9) Values from the current row are assigned to their corresponding targets identified by the <fetch target list>. The assignments are made in an implementation-dependent order. Let *TV* be a target and let *SV* denote its corresponding value in the current row of *CR*. The General Rules of Subclause 9.1, "Retrieval assignment", are applied to *TV* and *SV* as *TARGET* and *VALUE*, respectively.

10) If an exception condition occurs during the assignment of *SV* to *TV*, then the value of *TV* is implementation-dependent and *CR* remains positioned on the current row.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <fetch statement> shall not contain a <fetch orientation>.

   b) A <fetch statement> shall not specify FROM.

   c) If the data type of the target identified by the *i*-th <target specification> in the <fetch target list> is an exact numeric type, then the data type of the *i*-th column of the table *T* shall be an exact numeric type.

15.4 <close statement>

Function

Close a cursor.

Format

```
<close statement> ::=
       CLOSE <cursor name>
```

Syntax Rules

ANSI Only–SQL3

None.

ISO Only—caused by ANSI changes not yet considered by ISO

1) The containing <module> shall contain a <declare cursor> *CR* whose <cursor name> is the same as the <cursor name> of the <close statement>.

Access Rules

None.

General Rules

ANSI Only–SQL3

1) Let *CR* be the <declare cursor> denoted by the <cursor name>.

2) If cursor *CR* is not in the open state, then an exception condition is raised: *invalid cursor state*.

3) If the cursor mode of the current SQL-transaction is *cascade off*, and if some <fetch statement> in the current set-processing mode session of the current SQL-transaction has not resulted in a <status parameter> indicating a completion condition of *no data*, then an exception condition is raised: *invalid SQL statement*.

4) Cursor *CR* is placed in the closed state and the copy of the <cursor specification> of *CR* is destroyed.

5) Let *CC* be the set of constraints that are defined by <assertion definition>s or by <table constraint definition>s of tables that are nodes in the constraint-referential-action graph for *CR*. Let *CCL* be a comma-separated list of the <constraint name>s of the constraints in *CC*. The following <set constraints mode statement> is effectively executed:

   SET *CCL* CONSTRAINTS IMMEDIATE

   Note: *The constraint-referential-action graph is defined in Subclause 15.2, "<open statement>".*

6) If the cursor mode of the current SQL-transaction is *cascade off*, then the cursor mode of the current SQL-transaction is set to *cascade on*.

7) Any triggered actions that were deferred are effectively executed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

15.5 <select statement: single row>

Function

Retrieve values from a specified row of a table.

Format

```
<select statement: single row> ::=
       SELECT [ <set quantifier> ] <select list>
         INTO <select target list>
           <table expression>

<select target list> ::=
       <target specification> [ { <comma> <target specification> }... ]
```

Syntax Rules

1) The number of elements in the <select list> shall be the same as the number of elements in the <select target list>. The *i*-th <target specification> in the <select target list> corresponds with the *i*-th element of the <select list>.

2) The Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to each corresponding <target specification> and <value expression>, as *TARGET* and *VALUE*, respectively.

3) Let *S* be a <query specification> whose <select list> and <table expression> are those specified in the <select statement: single row> and that specifies the <set quantifier> if it is specified in the <select statement: single row>. *S* shall be a valid <query specification>.

Access Rules

None.

General Rules

1) Let *Q* be the result of <query specification> *S*.

2) Case:

   a) If the cardinality of *Q* is greater than 1, then an exception condition is raised: *cardinality violation*. It is implementation-dependent whether or not SQL-data values are assigned to the targets identified by the <select target list>.

   b) If *Q* is empty, then no SQL-data values are assigned to any targets identified by the <select target list>, and a completion condition is raised: *no data*.

   c) Otherwise, values in the row of *Q* are assigned to their corresponding targets.

3) If a completion condition *no data* has been raised, then no further General Rules of this Subclause are applied.

4) The assignment of values to targets in the <select target list> is in an implementation-dependent order.

5) If an exception condition is raised during the assignment of a value to a target, then the values of targets are implementation-dependent.

6) The target identified by the *i*-th <target specification> of the <select target list> corresponds to the *i*-th value in the row of *Q*.

7) Let *TV* be an identified target and let *SV* be its corresponding value in the row of *Q*.

8) The General Rules of Subclause 9.1, "Retrieval assignment", are applied to *TV* and *SV*, as *TARGET* and *VALUE*, respectively.

9) Any <column name> or <function name> contained in the <select list> shall have <encapsulation level> PUBLIC unless the <select statement: single row> is contained in a <table definition> or <abstract datatype definition> whose descriptor contains a column descriptor or function descriptor for that column name or function name.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If the data type of the target identified by the *i*-th <target specification> in the <select target list> is an exact numeric type, then the data type of the *i*-th column of the table *T* shall be an exact numeric type.

   b) The <table expression> shall not include a <group by clause> or a <having clause> and shall not identify a grouped view.

15.6 <delete statement: positioned>

Function

Delete a row of a table.

Format

```
<delete statement: positioned> ::=
       DELETE [ FROM <table name> ]
         WHERE CURRENT OF <cursor name>
```

****Editor's Note****

<delete statement: positioned> used "<table name>", while <delete statement: searched>, <insert statement>, <update statement: positioned>, and <update statement: searched> all use "<table reference>". See Language Opportunity **276** in the Editor's Notes.

Syntax Rules

ANSI Only–SQL3

1) Let *CR* be the <declare cursor> denoted by the <cursor name>.

ISO Only—caused by ANSI changes not yet considered by ISO

2) The containing <module> shall contain a <declare cursor> whose <cursor name> is the same as the <cursor name> in the <delete statement: positioned>. Let *CR* be the cursor specified by <cursor name>.

3) *CR* shall be an updatable cursor.

   Note: ''updatable cursor'' is defined in Subclause 15.1, "<declare cursor>".

4) Let *T* be the table identified by the <table name>. Let *QS* be the <query specification> that is the simply underlying table of the simply underlying table of *CR*. The simply underlying table of *QS* shall be *T*.

****Editor's Note****

In the Format, the <table name> is optional; however, rules like the preceding one, use the <table name> explicitly. While the semantics are well understood, the wording of the rules must be corrected. See Possible Problem **277** in the Editor's Notes.

Note: The ''simply underlying table'' of a <cursor specification> is defined in Subclause 15.1, "<declare cursor>".

• 1 Rule moved

5) If <table name> is specified, then <table name> shall identify the table denoted by *T*.

Access Rules

1) The applicable privileges for each <table name> in the first <from clause> in the <cursor specification> of *CR* shall include DELETE. If *T* is derived from a <joined table> that specifies FOREIGN KEY or CONSTRAINT and the <referential action> in the <referential constraint definition> specifies SET NULL or SET DEFAULT, then the applicable privileges for all columns in the <referencing columns> in that <referential constraint definition> shall include UPDATE.

   Note: The *applicable privileges* for a <table name> are defined in Subclause 10.5, "<privileges>".

General Rules

1) If the access mode of the current SQL-transaction is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state*.

2) If there is any cursor *CR* that is currently open and whose <declare cursor> specified SENSITIVE, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception—request failed*.

3) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception—request failed*.

4) The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

5) If cursor *CR* is not positioned on a row, then an exception condition is raised: *invalid cursor state*.

6) If the cursor mode of the current SQL-transaction is *cascade off*, and if a prior <SQL procedure statement> executed within the current set-processing mode session of the current SQL-transaction was an <update statement: positioned>, then an exception condition is raised: *invalid SQL statement*. If *T* is associated with an abstract data type *ADT*, then the destructor function for *ADT* is invoked with its <argument> being the row from which the current row of *CR* is derived.

7) The row from which the current row of *CR* is derived is marked for deletion.

8) If, while *CR* is open, the row from which the current row of *CR* is derived has been marked for deletion by any <delete statement: searched>, marked for deletion by any <delete statement: positioned> that identifies any cursor other than *CR*, updated by any <update statement: searched>, or updated by any <update statement: positioned> that identifies any cursor other than *CR*, then a completion condition is raised: *warning—cursor operation conflict*.

9) Case:

   a) If the cursor mode of the current SQL-transaction is *cascade on*, then all rows that are marked for deletion are effectively deleted at the end of the <delete statement: positioned> prior to the checking of any integrity constraint.

b) If the cursor mode of the current SQL-transaction is *cascade off*, then the deletion of all rows that are marked for deletion is effectively deferred until implicitly enacted by the execution of a <commit statement> or explicitly enacted by execution of a <close statement>.

10) If the <delete statement: positioned> deleted the last row of *CR*, then the position of *CR* is after the last row; otherwise, the position of *CR* is before the next row.

11) If the <delete statement: positioned> is the triggering statement for an <assertion definition> and the constraint mode of the assertion in the current SQL-transaction is *deferred*, then the <assertion definition> becomes a pending <assertion definition>.

12) If *T* belongs to a subtable family, then every row in the corresponding subtable row family is marked for deletion.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

15.7 <delete statement: searched>

Function

Delete rows of a table.

Format

```
<delete statement: searched> ::=
       DELETE FROM <table reference>
         [ WHERE <search condition> ]
```

Syntax Rules

1) Let *T* be the table identified by the <table reference>. *T* shall not be a read-only table. *T* shall not be a table derived from a <query expression> that specifies INTERSECT or EXCEPT.

ANSI Only–SQL3

Note: *T* may be any table, set, multiset or list.

- 1 Rule moved

2) The scope of the <table reference> is the entire <delete statement: searched>.

3) If the <delete statement: searched> is contained in a <triggered SQL statement>, then the <search condition> shall not contain a <value specification> that specifies a
ANSI parameter reference.
ISO <parameter specification>.

Access Rules

1) The applicable privileges for the <table name> in the <table reference> that is not in a <subquery> shall include DELETE unless the <delete statement: searched> is contained in a <triggered SQL statement>. If *T* is derived from a <joined table> that specifies FOREIGN KEY or CONSTRAINT and the <referential action> in the <referential constraint definition> specifies SET NULL or SET DEFAULT, then the applicable <privileges> for all columns in the <referencing columns> in that <referential constraint definition> shall include UPDATE.

Note: The *applicable privileges* for a <table name> are defined in Subclause 10.5, "<privileges>".

General Rules

1) If the access mode of the current SQL-transaction is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state*.

2) If there is any cursor *CR* that is currently open and whose <declare cursor> specified SENSITIVE, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception—request failed*.

3) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception—request failed*.

4) The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

5) Case:

   a) If <search condition> is not specified, then all rows of *T* are marked for deletion. For every row marked for deletion, if *T* is associated with an abstract data type *ADT*, then the destructor function for *ADT* is invoked with its <argument> being that row.

   b) If <search condition> is specified, then it is applied to each row of *T* with the <table reference> bound to that row, and all rows for which the result of the <search condition> is true are marked for deletion.

      The <search condition> is effectively evaluated for each row of *T* before marking for deletion any row of T.

      Each <subquery> in the <search condition> is effectively executed for each row of *T* and the results used in the application of the <search condition> to the given row of *T*. If any executed <subquery> contains an outer reference to a column of *T*, the reference is to the value of that column in the given row of *T*.

      For every row marked for deletion, if *T* is associated with an abstract data type *ADT*, then the destructor function for *ADT* is invoked with its <argument> being that row.

      ****Editor's Note****

      Although the destructor function for a type is invoked, there is no guarantee that a <destroy statement> will actually be executed and the object ADT instance destroyed. See Possible Problem 277 in the Editor's Notes.

      Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

   c) Any <column name> or <function name> contained in the <search condition> shall have encapsulation level PUBLIC unless the <delete statement: searched> is contained in a <table definition> or <abstract datatype definition> whose descriptor contains a column descriptor or function descriptor for that column name or function name.

6) If any row that is marked for deletion by the <delete statement: searched> has been marked for deletion by any <delete statement: positioned> that identifies some cursor *CR* that is still open or updated by any <update statement: positioned> that identifies some cursor *CR* that is still open, then a completion condition is raised: *warning—cursor operation conflict*.

7) All rows that are marked for deletion are effectively deleted at the end of the <delete statement: searched>, prior to the checking of any integrity constraints.

8) If <search condition> is specified, then the <search condition> is evaluated for each row of *T* prior to the invocation of any <triggered action> caused by the imminent or actual deletion of any row of *T*.

9) If the <delete statement: searched> is the triggering statement for an <assertion definition> and the constraint mode of the assertion in the current SQL-transaction is *deferred*, then the <assertion definition> becomes a pending <assertion definition>.

10) If *T* belongs to a subtable family, then every row in the corresponding subtable row family is marked for deletion.

11) If no row is deleted, then a completion condition is raised: *no data*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) No leaf generally underlying table of *T* shall be an underlying table of any <query expression> generally contained in the <search condition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

15.8 <insert statement>

Function

Create new rows in a table.

Format

```
<insert statement> ::=

        ANSI Only--SQL3
        INSERT INTO { <table reference> | CURSOR <cursor name> }
        ISO Only---caused by ANSI changes not yet considered by ISO
        INSERT INTO <table reference>

          <insert columns and source>
        ANSI Only--SQL3

          [ <insert point> ]

          [ ALIAS <target specification> ]

<insert columns and source> ::=
          [ <left paren> <insert column list> <right paren> ] <query expression>
        | DEFAULT VALUES

<insert column list> ::= <column name list>
ANSI Only--SQL3


<insert point> ::=
        <relative insert point> ELEMENT <where clause>

<relative insert point> ::=
          BEFORE
        | AFTER
```

Syntax Rules

ANSI Only–SQL3

1) If <cursor name> is specified, then <insert point> shall not be specified and the containing <module> shall contain a <declare cursor> *CR* whose <cursor name> is the same as the <cursor name> in the <insert statement>.

2) Case:

   a) If <cursor name> is specified, then let *T* be the underlying table of the <cursor specification>.

   b) Otherwise, let *T* be the table identified by the <table reference>.

3) *T* shall not be a read-only table or a table that is identified in any <from clause> of the <query expression>. *T* shall not be a table derived from a <table expression> that specifies a set operator UNION or EXCEPT.

ISO Only—caused by ANSI changes not yet considered by ISO

4) Let *T* be the table identified by the <table reference>. *T* shall not be a read-only table or a table that is identified in any <from clause> of the <query expression>. *T* shall not be a table derived from a <table expression> that specifies a set operator UNION or EXCEPT.

5) If <insert point> is specified, then *T* shall be of <table type> LIST.

6) An <insert columns and source> that specifies DEFAULT VALUES is equivalent to an <insert columns and source> that specifies a <query expression> of the form

   VALUES (DEFAULT, . . . , DEFAULT)

   where the number of ''DEFAULT'' entries is equal to the number of columns of *T*.

7) No <column name> of T shall be identified more than once. If <insert columns and source> is specified and the <insert column list> is omitted, then an <insert column list> that identifies all updatable columns of *T* with <encapsulation level> PUBLIC in the ascending sequence of their ordinal positions within *T* is implicit.

8) If the <insert statement> is not contained in a <triggered SQL statement>, then

   Case:

   a) If an <insert column list> is specified, then the applicable privileges for each <table name> contained in the <table reference> that is not in a <subquery> shall include INSERT for each <column name> in the <insert column list>.

      Note: The *applicable privileges* for a <table name> are defined in Subclause 10.5, "<privileges>".

   b) Any <column name> or <function name> contained in the <column name list> shall have encapsulation level PUBLIC unless the <insert statement> is contained in a <table definition> or <abstract datatype definition> whose descriptor contains a column descriptor or function descriptor for that column name or function name.

c) If *T* is derived from a <joined table> that specifies FOREIGN KEY or CONSTRAINT in the <join condition> and LEFT, RIGHT, or FULL in the <join type>, then the applicable privileges for the table identified by that <joined table> shall include UPDATE for all columns in the <referencing columns> in the <referential constraint definition>.

Note: The *applicable privileges* for a <table name> are defined in Subclause 10.5, "<privileges>".

9) A column identified by the <insert column list> is an object column.

10) An object column shall be an updatable or constant column.

11) If ALIAS is specified, then *T* shall contain a <column definition> with <column name> OID.

12) Let *QT* be the table specified by the <query expression>. The degree of *QT* shall be equal to the number of <column name>s in the <insert column list>. The column of table *T* identified by the *i*-th <column name> in the <insert column list> corresponds with the *i*-th column of *QT*.

13) The Syntax Rules of Subclause 9.2, "Store assignment", apply to corresponding columns of *T* and *QT* as *TARGET* and *VALUE*, respectively.

14) If the <insert statement> is contained in a <triggered SQL statement>, then the insert value shall not contain a <value specification> that specifies a
ANSI parameter reference.
ISO <parameter specification>.

Access Rules

1) Case:

a) If an <insert column list> is specified, then the applicable privileges shall include INSERT for each <column name> in the <insert column list>.

b) Otherwise, the applicable privileges shall include INSERT for each <column name> in *T*.

Note: The *applicable privileges* for a <table name> are defined in Subclause 10.5, "<privileges>".

2) Each <column name> in the <insert column list> shall identify a column of *T*.

General Rules

1) If the access mode of the current SQL-transaction is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state*.

2) If there is any open cursor *CR* in the current SQL-transaction that was opened CASCADE OFF and whose constraint-referential-action graph is not distinct from the constraint-referential-action graph of the <insert statement>, then an exception condition is raised: *invalid SQL statement*.

Note: *The constraint-referential-action graph is defined in Subclause 15.2, "<open statement>".*

3) If there is any cursor *CR* that is currently open and whose <declare cursor> specified SENSITIVE, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception—request failed*.

4) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception—request failed*.

5) The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

ANSI Only–SQL3

6) If *T* is of <table type> LIST and <cursor name> is not specified, then:

Case:

a) If <insert point> is specified, then:

Case:

i) If the result of the <search condition> immediately contained in the <where clause> is true for one and only one row of *T*, then any rows to be inserted BEFORE that row will be inserted in order with the last insertion immediately preceding that row; any rows to be inserted AFTER that row will be inserted in order with the first insertion immediately following that row.

ii) Otherwise, an exception condition is raised: *data exception—insertion point not unique*.

b) Otherwise, any rows to be inserted will be inserted in order following the last row of the table, if any, or beginning at position 1 if the table is empty.

7) If <insert columns and source> is specified and immediately contains a <query expression>, then let *Q* be the result of that <query expression>. If *Q* is empty, then no row shall be inserted and a completion condition is raised: *no data*. If ALIAS is specified and the <query expression> produces more than one candidate row, then no row is inserted and an exception condition is raised: *cardinality violation*.

8) Let *B* be the leaf generally underlying table of *T*.

****Editor's Note****

Possible Problem 250 in the Editor's Notes identifies a concern over the use of ''leaf generally underlying table'' here; in SQL2, an updatable view could have only one leaf generally underlying table, but in SQL3, more views are updatable and there may be more than one leaf generally underlying table involved.

9) The <query expression> is effectively evaluated before inserting any rows into *B*.

10) Let *Q* be the result of that <query expression>.

Case:

a) If *Q* is empty, then no row is inserted and a completion condition is raised: *no data*.

b) Otherwise, for each row *R* of *Q*:

i) A candidate row of *B* is effectively created in which the value of each column is its default value, as specified in the General Rules of Subclause 11.5, "<default clause>". The candidate row includes every stored column of *B*.

ii) If <insert columns and source> immediately contains a <query expression>, then each object column in the candidate row is updated and set equal to the corresponding insert value.

Note: For a virtual column, the update will invoke the corresponding set function, as defined in the General Rules of Subclause 15.9, "<update statement: positioned>".

iii) If *T* contains a default constructor function, then invoke that function with each parameter *P* consisting of the value in the candidate row of the column with the same name as *P*.

iv) For every object column in the candidate row, the value of the object column identified by the *i*-th <column name> in the <insert column list> is replaced by the *i*-th value of *R*.

v) If the candidate row has a column named OID whose value is a null value, then assign to it a unique object identifier value.

vi) Let *C* be a column that is represented in the candidate row and let *SV* be its value in the candidate row. The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *VALUE*, respectively.

ANSI Only–SQL3

vii) If *T* is a list and <cursor name> is specified, then

1) If *CR* is positioned on a row, then the candidate row is inserted after that row and *CR* is positioned on the new row.

2) If *CR* is positioned before a row or after the last row, then the candidate row is inserted at that position and *CR* is positioned on the new row.

viii) Case:

1) If *T* is a set and the candidate row is equal to an existing row in the table, then a completion condition is raised: *warning—duplicate element eliminated in set insertion*.

2) Otherwise, the candidate row is inserted into the corresponding stored columns of *B*.

ISO Only—caused by ANSI changes not yet considered by ISO

ix) The candidate row is inserted into the corresponding stored columns of *B*.

Note: The data values allowable in the candidate row may be constrained by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 11.22, "<view definition>".

ISO Only—caused by ANSI changes not yet considered by ISO

11) If <insert columns and source> immediately contains a <query expression>, then let *R* denote a row of *Q*. Otherwise, let *R* denote the candidate row effectively created as specified in the General Rules of Subclause 11.5, "<default clause>". The insert values of one candidate row are the values in one row of *Q* and the values in one row of *Q* are the insert values of one candidate row.

12) Let *R* be a row of *Q*. The *i*-th value of *R* is the insert value of the object column identified by the *i*-th <column name> in the <insert column list>.

13) Let *C* be an object column and let *SV* be an insert value of *C*.

14) The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *VALUE*, respectively.

15) If the <insert statement> is the triggering statement for an <assertion definition> and the constraint mode of the assertion in the current SQL-transaction is *deferred*, then the <assertion definition> becomes a pending <assertion definition>.

16) If ALIAS is specified, then let *SV* be the object identifier of the newly inserted object and let *TV* be the <target specification>. The General Rules of Subclause 9.1, "Retrieval assignment", are applied to *TV* and *SV*, as *TARGET* and *VALUE*, respectively.

17) If *T* is a subtable, then for each supertable of *T*,

Case:

a) If there is a row of that supertable that has the same primary key value as the candidate row, then an exception condition is raised: *data exception—object already exists*.

b) If there is no row of that supertable that has the same primary key value as the candidate row, then a row is inserted into that supertable. Values of the row inserted into the supertable are the same as the values of the corresponding inherited columns in the candidate row.

Note: If *T* is a maximal supertable, then a row is inserted only into *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

• 1 Rule deleted

a) An <insert statement> shall not specify ALIAS.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) The leaf generally underlying table of *T* shall not be generally contained in the <query expression> immediately contained in the <insert columns and source> except as the <qualifier> of a <column reference>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) The <query expression> that is contained in an <insert columns and source> shall be a <query specification> or it shall be a <table value constructor> that contains exactly one <row value constructor> of the form ''<left paren> <row value constructor list> <right paren>'', and each <row value constructor element> of that <row value constructor list> shall be a <value specification>.

b) If the data type of the target identified by the *i*-th <column name> is an exact numeric type, then the data type of the *i*-th item of the <insert statement> shall be an exact numeric type.

c) If the data type of the target *C* identified by the *i*-th <column name> is character string, then the length in characters of the *i*-th item of the <insert statement> shall be less than or equal to the length of *C*.

d) The <insert columns and source> shall immediately contain a <query expression>.

15.9 <update statement: positioned>

Function

Update a row of a table.

Format

```
<update statement: positioned> ::=
       UPDATE [ <table reference> ]
         SET [ <update type> ] <set clause list>
           WHERE CURRENT OF <cursor name>

<set clause list> ::=
       <set clause> [ { <comma> <set clause> }... ]

<set clause> ::=
       <update target> <equals operator> <row value constructor>

<update target> ::=
         <object column>
       | <left paren> <object column list> <right paren>

<object column list> ::=
       <object column> [ { <comma> <object column> }... ]

<update type> ::= ALL | SOME | NONE

<object column> ::= <column name>
```

Syntax Rules

1) If <update type> is not specified, then an <update type> of ALL is implicit.

ANSI Only–SQL3

2) Let *CR* be the <declare cursor> denoted by the <cursor name>.

ISO Only—caused by ANSI changes not yet considered by ISO

3) The containing <module> shall contain a <declare cursor> for a cursor whose <cursor name> is the same as the <cursor name> in the <update statement: positioned>. Let *CR* be the cursor specified by <cursor name>.

4) *CR* shall be an updatable cursor.
Note: ''updatable cursor'' is defined in Subclause 15.1, "<declare cursor>".

5) Let *T* be the table identified by the <table reference>. Let *QS* be the <query specification> that is the simply underlying table of the simply underlying table of *CR*. The simply underlying table of *QS* shall be *T*.

Note: The ''simply underlying table'' of a <cursor specification> is defined in Subclause 15.1, "<declare cursor>".

6) If *CR* is an ordered cursor, then for each <object column> *OC*, the column of *T* identified by *OC* shall not be directly or indirectly referenced in the <order by clause> of the defining <cursor specification> for *CR*.

7) No leaf generally underlying table of *T* shall be an underlying table of any <query expression> generally contained in any <value expression> immediately contained in any <update source> contained in the <set clause list>.

8) A <value expression> in a <set clause> shall not directly contain a <set function specification>.

9) The same <object column> shall not appear more than once in a <set clause list>.

10) Every column identified by an <object column> shall be an updatable column.

11) Any <column name> or <function name> contained in the <set clause list> shall have encapsulation level PUBLIC unless the <update statement: positioned> is contained in a <table definition> or <abstract datatype definition> whose descriptor contains a column descriptor or function descriptor for that column name or function name.

12) For every <column name> that identifies a virtual column, there shall be a set function associated with that column.

13) If the cursor identified by <cursor name> was specified using an explicit or implicit <updatability clause> of FOR UPDATE, then each <column name> specified as an <object column> shall identify a column in the explicit or implicit <column name list> associated with the <updatability clause>.

14) If *T* is derived from a <joined table> that specifies FOREIGN KEY or CONSTRAINT, then each <column name> specified as an <object column> shall not identify a column of a referenced table of that <joined table>.

15) The scope of the <table reference> is the entire <update statement: positioned>.

16) For every <object column> in a <set clause>, if that <object column> identifies a stored column, then Syntax Rules of Subclause 9.2, "Store assignment", apply to the column of *T* identified by the <object column> and the <row value constructor> of the <set clause> as *TARGET* and *VALUE*, respectively.

17) A <null state> shall identify a null state in the null class of the column identified by <object column>.

18) The <set clause> shall not contain a <column name> of OID.

Access Rules

1) The applicable privileges shall include UPDATE for each <object column>.

2) If *T* is derived from a <joined table> that specifies LEFT, RIGHT, or FULL, then the applicable privileges for each <table name> in that <joined table> that is not in a <subquery> shall include INSERT.

   Note: The *applicable privileges* for a <table name> are defined in Subclause 10.5, "<privileges>".

3) Each <column name> specified as an <object column> shall identify a column of *T*.

General Rules

1) If the <open statement> that opened *CR* specified CASCADE OFF, then the values of USER and any <datetime value function> generally contained in <set clause> shall be those that would have resulted at the time of the execution of the <open statement> which opened *CR*.

2) If the access mode of the current SQL-transaction is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state*.

3) If there is any cursor *CR* that is currently open and whose <declare cursor> specified SENSITIVE, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception—request failed*.

4) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception—request failed*.

5) The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

6) If the cursor mode of the current SQL-transaction is *cascade off*, then if the <update statement: positioned> is different from any <update statement: positioned> that had previously been executed in the current set-processing mode session of the current SQL-transaction, or if a <delete statement: positioned> had previously been executed in the current set-processing mode session of the current SQL-transaction, then an exception condition is raised: *invalid SQL statement*.

7) If the cursor mode of the current SQL-transaction is *cascade off*, then if the value of some <simple target specification> that is contained in the <set clause> has changed its value since the last time an <update statement: positioned> was last executed in the current set-processing mode session of the current SQL-transaction, then an exception condition is raised: *invalid target specification value*.

8) If the cursor mode of the current SQL-transaction is *cascade off*, and if the previous <SQL procedure statement> executed within the current set-processing mode session of the current SQL-transaction was an <update statement: positioned>, then an exception condition is raised: *invalid SQL statement*.

9) If cursor *CR* is not positioned on a row, then an exception condition is raised: *invalid cursor state*.

10) The object row is that row from which the current row of *CR* is derived.

11) If, while *CR* is open, the object row has been marked for deletion by any <delete statement: searched>, marked for deletion by any <delete statement: positioned> that identifies any cursor other than *CR*, updated by any <update statement: searched>, or updated by any <update statement: positioned> that identifies any cursor other than *CR*, then a completion condition is raised: *warning—cursor operation conflict*.

12) The value associated with DEFAULT is the default value for the <object column> in the containing <set clause>, as indicated in the General Rules of Subclause 11.5, "<default clause>".

13) The <value expression>s are effectively evaluated before updating the object row. If a <value expression> contains a reference to a column of
ANSI *T* or to *T* itself,
ISO *T*,
then the reference is to the value of that column
ANSI or table
in the object row before any value of the object row is updated.

14) *CR* remains positioned on its current row, even if an exception condition is raised during derivation of any <value expression> associated with the object row.

15) The update value is the value specified by <row value constructor>. If a <query expression> in a <row value constructor> is specified and the result of the <query expression> is empty, then

Case:

a) If ALL is specified, then the result of the <row value constructor> shall be *D* general null values, where *D* is the degree of the <query expression>.

b) If SOME or NONE is specified, then no object column shall be updated by the result of the <row value constructor>.

16) If a <query expression> in a <row value constructor> is specified, then:

Case:

a) If ALL or SOME is specified, then for each <set clause>, the value of the specified object column will be replaced by the specified update value.

b) If NONE is specified and the result of the <query expression> is empty, then the object row will not be changed.

17) A <set clause> specifies an object column and an update value of that column. The object column is the column identified by the <object column> in the <set clause>. The update value is the value specified by the <update source>.

Note: The data values allowable in the object row may be constrained by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 11.22, "<view definition>".

18) The object row is updated as specified by each <set clause>. For each <set clause>,

Case:

a) If the <object column> is a stored column, then the value of the specified object column, denoted by *C*, is replaced by the specified update value, denoted by *SV*. The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *VALUE*, respectively.

b) If the <object column> is a virtual column, then let *SF* be the set function associated with the virtual column, let *OR* be the object row, and let *SV* be the specified update value. Perform the function invocation *SF*(*OR*,*SV*).

19) If the <update statement: positioned> is the triggering statement for an <assertion definition> and the constraint mode of the assertion in the current SQL-transaction is *deferred*, then the <assertion definition> becomes a pending <assertion definition>.

20) If *T* is a supertable, then every row of the direct and the indirect subtables of *T* in the same subtable row family is changed such that all column values inherited from *T* replicate those of *T*

21) If *T* is a subtable and the <column name> of the <object column> specifies an inherited column, then every row in the corresponding subtable row family is changed such that all column values inherited by *T* replicate those of *T*

****Editor's Note****
The previous Rule makes no sense to the Editor. It appears to say that subtable T inherits values that are duplicates of the values in itself (T).

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <update statement: positioned> shall contain a <table name>.

   b) An <update statement: positioned> shall not contain an <update type>.

   c) An <update target> shall not contain an <object column list>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) *CR* shall not be an ordered cursor.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If the data type of the column identified by the *i*-th <object column> is an exact numeric type, then the data type of the *i*-th <value expression> in the <update statement: positioned> shall be an exact numeric type.

   b) If the data type of the column identified by the *i*-th <object column> *C* is character string, then the length in characters of the *i*-th <value expression> in the <update statement: positioned> shall be less than or equal to the length of *C*.

   c) An <update source> shall not specify DEFAULT.

15.10 <update statement: searched>

Function

Update rows of a table.

Format

```
<update statement: searched> ::=
       UPDATE <table reference>
```

ANSI Only--SQL3

```
         <update mechanism>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
         SET [ <update type> ] <set clause list>

         [ WHERE <search condition> ]
```

ANSI Only--SQL3

```
<update mechanism> ::=
          <update by setting>
        | <update by moving>

<update by setting> ::=
       SET [ <update type> ] <set clause list>

<update by moving> ::=
       MOVE <insert point>
```

Syntax Rules

1) If <update type> is not specified, then an <update type> of ALL is implicit.

ANSI Only–SQL3

2) If <update by moving> is specified, then *T* shall be of <table type> LIST.

3) Let *T* be the table identified by the <table reference>. *T* shall be an updatable table.

• 1 Rule moved

4) A <value expression> in a <set clause> shall not directly contain a <set function specification>.

5) The same <object column> shall not appear more than once in a <set clause list>.

6) Every column identified by an <object column> shall be an updatable column.

7) For every <column name> that identifies a virtual column thare shall be a set function associated with the column.

8) If *T* is derived from a <joined table> which specifies FOREIGN KEY or CONSTRAINT, then no <column name> specified as an <object column> shall identify a column of the referenced table of that <joined table>.

9) The scope of the <table reference> is the entire <update statement: searched>.

10) For every <object column> in a <set clause>, if that <object column> is a stored column, then Syntax Rules of Subclause 9.2, "Store assignment", apply to the column of *T* identified by the <object column> and the <value expression> of the <set clause> as *TARGET* and *VALUE*, respectively.

11) Any <column name> or <function name> contained in the <set clause list> or the <search condition> shall have encapsulation level PUBLIC unless the <update statement: searched> is contained in a <table definition> or <abstract datatype definition> whose descriptor contains a column descriptor or function descriptor for that column name or function name.

12) A <null state> shall identify a null state in the null class of the column identified by <object column>.

13) If the <update statement: searched> is contained in a <triggered SQL statement>, then the <search condition> shall not contain a <value specification> that specifies a
ANSI parameter reference.
ISO <parameter specification>.

Access Rules

1) The applicable privileges for each <table name> in <table reference> shall include UPDATE for each <object column> unless the <update statement: searched> is contained in a <triggered SQL statement>. If *T* is derived from a <joined table> that specifies LEFT, RIGHT, or FULL, then the applicable privileges for each <table name> in that <joined table> that is not in a <subquery> shall include INSERT for all columns unless the <update statement: searched> is contained in a <triggered SQL statement>.

Note: The *applicable privileges* for a <table name> are defined in Subclause 10.5, "<privileges>".

2) Each <column name> specified as an <object column> shall identify a column of *T*.

General Rules

1) If the access mode of the current SQL-transaction is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state*.

2) If there is any open cursor *CR* in the current SQL-transaction that was opened CASCADE OFF and whose constraint-referential-action graph is not disjoint from the constraint-referential-action graph of the <update statement: searched>, then an exception condition is raised: *invalid SQL statement*.

****Editor's Note****
The preceding rules deals with CASCADE ON and OFF and depend on the constraint-referential-action graph, which has been deleted. Therefore, these rules will need to be revised.

3) If there is any cursor *CR* that is currently open and whose <declare cursor> specified SENSITIVE, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception—request failed*.

4) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception—request failed*.

5) The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

6) Case:

   a) If a <search condition> is not specified, then all rows of *T* are the object rows.

   b) If a <search condition> is specified, then it is applied to each row of *T* with the <table name> bound to that row, and the object rows are those rows for which the result of the <search condition> is true. The <search condition> is effectively evaluated for each row of *T* before updating any row of *T*.

      Each <subquery> in the <search condition> is effectively executed for each row of *T* and the results used in the application of the <search condition> to the given row of *T*. If any executed <subquery> contains an outer reference to a column of
      ANSI *T* or to *T* itself,
      ISO *T*,
      then the reference is to the value of that column in the given row of *T*.

      Note: *Outer reference* is defined in Subclause 6.6, "<column reference> and <object reference>".

7) If any row in the set of object rows has been marked for deletion by any <delete statement: positioned> that identifies some cursor *CR* that is still open or updated by any <update statement: positioned> that identifies some cursor *CR* that is still open, then a completion condition is raised: *warning—cursor operation conflict*.

8) If a <search condition> is specified, then the <search condition> is evaluated for each row of *T* prior to the invocation of any <triggered action> caused by the update of any row of *T*.

• 1 (redundant) Rule deleted

9) The update value is the value specified by <row value constructor>. If a <query expression> in a <row value constructor> is specified and the result of the <query expression> is empty, then

   Case:

   a) If ALL is specified, then the result of the <row value constructor> shall be *D* general null values, where *D* is the degree of the <query expression>.

   b) If SOME or NONE is specified, then no object column shall be updated by the result of the <row value constructor>.

10) If a <query expression> in a <row value constructor> is specified, then:

Case:

a) If ALL or SOME is specified, then for each <set clause>, the value of the specified object column is replaced by the specified update value.

b) If NONE is specified and the result of the <query expression> is empty, then the object row is not changed.

11) If the set of object rows is empty, then a completion condition is raised: *no data*.

12) If a completion condition *no data* has been raised, then no further General Rules of this Subclause are applied.

13) The <value expression>s are effectively evaluated for each row of *T* before updating any row of *T*.

14) A <set clause> specifies an object column an an update value of that column. The object column is the column identified by the <object column> in the <set clause>. The update value is the value specified by the <update source>.

Note: The data values allowable in the object row may be constrained by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 11.22, "<view definition>".

ANSI Only–SQL3

15) Case:

a) If <update by setting> is specified, then the object row is updated.

i) If the <object column> is a stored column, then the value of the specified object column, denoted by *C*, is replaced by the specified update value, denoted by *SV*. The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *VALUE*, respectively.

ii) If the <object column> is a virtual column, then let *SF* be the set function associated with the virtual column, let *OR* be the object row, and let *SV* be the specified update value. Perform the function invocation *SF*(*OR*,*SV*).

b) If <update by moving> is specified, then:

Case:

i) If the result of the <search condition> immediately contained in the <where clause> immediately contained in the <insert point> is true for one and only one row of *T*, then an ordered set of rows to be moved BEFORE that row will be inserted in order with the last insertion immediately preceding that row; an ordered set of rows to be moved AFTER that row will be inserted in order with the first insertion immediately following that row.

ii) Otherwise, an exception condition is raised: *data exception—insertion point not unique*.

ISO Only—caused by ANSI changes not yet considered by ISO

16) The object row is updated as specified by each <set clause>. For each <set clause>,

Case:

a) If the <object column> is a stored column, then the value of the specified object column, denoted by *C*, is replaced by the specified update value, denoted by *SV*. The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *VALUE*, respectively.

b) If the <object column> is a virtual column, then let *SF* be the set function associated with the virtual column, let *OR* be the object row, and let *SV* be the specified update value. Perform the function invocation *SF*(*OR*,*SV*).

17) If the <update statement: searched> is the triggering statement for an <assertion definition> and the constraint mode of the assertion in the current SQL-transaction is *deferred*, then the <assertion definition> becomes a pending <assertion definition>.

18) If *T* is a supertable, then every row of the direct and the indirect subtables of *T* in the same subtable row family is changed such that all column values inherited from *T* replicate those of *T*

19) If *T* is a subtable and the <column name> of the <object column> specifies an inherited column, then every row in the corresponding subtable row family is changed such that all column values inherited by *T* replicate those of *T*

****Editor's Note****

The previous Rule makes no sense to the Editor. It appears to say that subtable T inherits values that are duplicates of the values in itself (T).

Leveling Rules

1) The following restrictions apply for Full SQL:

a) An <update statement: searched> shall not contain an <update type>.

b) An <update target> shall not contain an <object column list>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) No leaf generally underlying table of *T* shall be an underlying table of any <query expression> generally contained in the <search condition> or in any <value expression> immediately contained in any <update source> contained in the <set clause list>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) If the data type of the column identified by the *i*-th <object column> is an exact numeric type, then the data type of the *i*-th <value expression> in the <update statement: searched> shall be an exact numeric type.

b) If the data type of the column identified by the *i*-th <object column> *C* is character string, then the length in characters of the *i*-th <value expression> in the <update statement: searched> shall be less than or equal to the length of *C*.

15.11 <temporary table declaration>

Function

ANSI Only–SQL3

Declare a declared local temporary table.

ISO Only—caused by ANSI changes not yet considered by ISO

Declare a declared local temporary table that will be effectively materialized the first time that any <routine> in the <module> that contains the <temporary table declaration> is executed and whose scope is all the <routine>s of that <module> executed within the same SQL-session.

Format

```
<temporary table declaration> ::=
```

ANSI Only--SQL3

```
DECLARE LOCAL TEMPORARY <table type> <table name>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
DECLARE LOCAL TEMPORARY TABLE <qualified local table name>
```

```
  <table element list>
  [ ON COMMIT { PRESERVE | DELETE } ROWS ]
```

Syntax Rules

ANSI Only–SQL3

1) Let *TN* be the <table name> of a <temporary table declaration> *TTD*.

   Case:

   a) If *TN* contains a <local or schema qualifier> *LSQ*, then:

      i) If *TTD* is contained in a <compound statement>, then *LSQ* shall be the same as the <beginning label> of the innermost <compound statement> that contains *TTD*.

ii) Otherwise, *TTD* shall be contained in a <module> *M* and *LSQ* shall be either ''MODULE'' or the <module name> of *M*.

b) If *TN* does not contain a <local or schema qualifier>, then:

Case:

i) If *TTD* is contained in a <compound statement>, then the <beginning label> of the innermost <compound statement> that contains *TTD* is implicit.

ii) Otherwise, *TTD* shall be contained in a <module> *M* and ''MODULE'' is implicit.

ISO Only—caused by ANSI changes not yet considered by ISO

2) Let *T* be the <local table name> of <qualified local table name>. *T* shall be different from the <local table name> of any other <temporary table declaration> or <temporary view definition> contained within the <module>.

3) Let *A* be the current <authorization identifier>.

4) If LIKE is specified, then:

a) Let *T1* be the table identified in the <like clause>.

b) The <like clause> is effectively replaced by *columns* where *columns* contains a <column definition> for every column that is defined in *T1*, in the order in which the columns appear in *T1*.

Note: <column constraint>s are not included in *columns*; <column constraint>s are effectively transformed to <table constraints> and are thereby excluded.

5) The descriptor of the table defined by a <temporary table declaration> includes the name of *T* and the column descriptor specified by each <column definition>. The *i*-th column descriptor is given by the *i*-th <column definition>.

6) A <temporary table declaration> shall contain at least one <column definition>.

7) If ON COMMIT is not specified, then ON COMMIT DELETE ROWS is implicit.

Access Rules

1) *A* shall have SELECT privilege on *T1*.

General Rules

1) Let *U* be the implementation-dependent <schema name> that is effectively derived from the implementation-dependent SQL-session identifier associated with the SQL-session and an implementation-dependent name associated with the <module> that contains the <temporary table declaration>.

2) The definition of *T* within a module is effectively equivalent to the definition of a persistent base table *U.T*. Within the module, any reference to MODULE.*T* is equivalent to a reference to *U.T*.

3) A set of privilege descriptors is created that define the privileges INSERT, SELECT, UPDATE, DELETE, and REFERENCES on this table and INSERT (<column name>), UPDATE (<column name>), and REFERENCES (<column name>) for every <column definition> in the table definition to *A*. These privileges are not grantable. The grantor for each of these privilege descriptors is set to the special grantor value ''_SYSTEM''.

4) The definition of a temporary table persists for the duration of the SQL-session. The termination of the SQL-session is effectively followed by the execution of the following <drop table statement> with the current <authorization identifier> and current <schema name> *U* without further Access Rule checking:

   DROP TABLE *T*

5) The definition of a declared local temporary table does not appear in any view of the Information Schema.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <temporary table declaration>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

15.12 <temporary view definition>

Function

Define a temporary viewed table whose definition exists for the duration of the current SQL-session.

Format

```
<temporary view definition> ::=
       CREATE TEMPORARY VIEW <table name>
           [ <left paren> <view column list> <right paren> ]
           AS <query expression>
```

Syntax Rules

1) A <table name> shall not specify a <schema name>. The implied <schema name> *U* is the unique implementation-defined <schema name> for the invocation of the <routine>s in the containing <module>.

2) Let *T* be the <qualified identifier> of <table name>.

3) *T* shall be different from the <qualified identifier> of any other <temporary view definition> or <temporary table declaration> contained within the <module>.

4) *T* shall be different from the <qualified identifier> of any other <table definition> or <view definition> in any schema whose <schema name> is identical to the <module authorization identifier> of the containing <module>.

5) If the <query expression> is updatable, then the temporary viewed table is an updatable table. Otherwise, it is a read-only table.

6) If any two columns in the table specified by the <query expression> have the same <column name>, or if any column of that table has an implementation-dependent name, then a <view column list> shall be specified.

7) The same <column name> shall not be specified more than once in the <view column list>.

8) The number of <column name>s in the <view column list> shall be the same as the degree of the table specified by the <query expression>.

Access Rules

None.

General Rules

1) A <temporary view definition> defines a temporary viewed table. The temporary viewed table, *V*, is the table that would result if the <query expression> were evaluated. Whether a temporary viewed table is materialized is implementation-dependent.

2) If the definition of a temporary view has not been dropped before the end of the SQL-session, then the definition of a temporary view persists for the duration of the SQL-session. The termination of the SQL-session is effectively followed by an <SQL schema statement> in a <module> with <module authorization identifier> *U* that specifies DROP VIEW *T*.

3) A view descriptor is created that describes the temporary viewed table *V* defined by the <temporary view definition>. The view descriptor includes the <table name>, the <query expression>, and column descriptors taken from the tables specified by the <query expression>. If a <view column list> is specified, then the <column name> of the *i*-th column of the view is the *i*-th <column name> in that <view column list>. Otherwise, the <column name>s of the view are the <column name>s of the table specified by the <query expression>.

4) Let *A* be the <authorization identifier> of the current SQL-session.

5) A set of privilege descriptors is created that defines the SELECT privilege on *V* to *A* and SELECT for each column of *V* to *A*. These privileges are not grantable. The grantor for each of these privileges is set to the special grantor value ''_SYSTEM''.

6) If *V* is updatable, then let *UT* be the underlying table of the <query expression>.

7) If *V* is updatable, then:

   a) A set of privilege descriptors is created that defines the privileges INSERT, UPDATE, and DELETE on *V* that are the applicable privileges on *UT* to *A*. These privilegesare not grantable. The grantor for each of these privilege descriptors is set to the special grantor value ''_SYSTEM''.

   b) For every column in *V*:

      i) There is a corresponding column in *UT* from which the column of *V* is derived. Let *CV* and *CT* be the column names of *V* and *T* respectively.

      ii) A set of privilege descriptors is created that defines the privileges INSERT(*CV*) and UPDATE(*CV*) on *V* that correspond to the privileges INSERT(*CT*) and UPDATE(*CT*) on *UT*. These privileges are not grantable. The grantor for each of these privilege descriptors is set to the special grantor value ''_SYSTEM''.

   c) If *V* is updatable, then for each row in *V* there is a corresponding row in *UT* from which the row of *V* is derived and for each column in *V* there is a corresponding column in *UT* from which the column of *V* is derived. The insertion of a row into *V* is an insertion of a corresponding row into *UT*. The deletion of a row from *V* is a deletion of the corresponding row in *UT*. The updating of a column of a row in *V* is an updating of the corresponding column of the corresponding row in *UT*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <temporary view definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

16 Transaction management

16.1 <set transaction statement>

Function

Set the attributes of the next SQL-transaction for the SQL-agent.

Format

```
<set transaction statement> ::=
       SET TRANSACTION <transaction mode> [ { <comma> <transaction mode> }... ]

<transaction mode> ::=
         <isolation level>
       | <transaction access mode>
       | <diagnostics size>

<transaction access mode> ::=
         READ ONLY
       | READ WRITE

<isolation level> ::=
       ISOLATION LEVEL <level of isolation>

<level of isolation> ::=
         READ UNCOMMITTED
       | READ COMMITTED
       | REPEATABLE READ
       | SERIALIZABLE

<diagnostics size> ::=
       DIAGNOSTICS SIZE <number of conditions>

<number of conditions> ::= <simple value specification>
```

Syntax Rules

1) No <transaction mode> shall be specified more than once.

2) If an <isolation level> is not specified, then a <level of isolation> of ISOLATION LEVEL SERIALIZABLE is implicit.

3) If READ WRITE is specified, then the <level of isolation> shall be not be READ UNCOMMITTED.

4) If a <transaction access mode> is not specified and a <level of isolation> of READ UNCOMMITTED is specified, then READ ONLY is implicit. Otherwise, READ WRITE is implicit.

5) The data type of <number of conditions> shall be exact numeric with scale 0.

Access Rules

None.

General Rules

1) If a <set transaction statement> statement is executed when an SQL-transaction is currently active, then an exception condition is raised: *invalid transaction state*.

2) If <number of conditions> is specified and is less than 1, then an exception condition is raised: *invalid condition number*.

3) Let *TXN* be the next SQL-transaction for the SQL-agent.

4) If READ ONLY is specified, then the access mode of *TXN* is set to *read-only*. If READ WRITE is specified, then the access mode of *TXN* is set to *read-write*.

5) The isolation level of *TXN* is set to an implementation-defined isolation level that will not exhibit any of the phenomena that the explicit or implicit <level of isolation> would not exhibit, as specified in Table 9, "SQL-transaction isolation levels and the three phenomena".

6) If <number of conditions> is specified, then the diagnostics area limit of *TXN* is set to <number of conditions>.

7) If <number of conditions> is not specified, then the diagnostics area limit of *TXN* is set to an implementation-dependent value not less than 1.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <set transaction statement>.

16.2 <set constraints mode statement>

Function

If an SQL-transaction is currently active, then set the constraint mode for that SQL-transaction in the current SQL-session. If no SQL-transaction is currently active, then set the constraint mode for the next SQL-transaction in the current SQL-session for the SQL-agent.

Format

```
<set constraints mode statement> ::=
       SET CONSTRAINTS <constraint name list> { DEFERRED | IMMEDIATE }

<constraint name list> ::=
         ALL
       | <constraint name> [ { <comma> <constraint name> }... ]
```

Syntax Rules

1) If a <constraint name> is specified, then it shall identify a constraint.

2) The constraint identified by <constraint name> shall be DEFERRABLE.

Access Rules

None.

General Rules

1) If an SQL-transaction is currently active, then let *TXN* be the currently active SQL-transaction. Otherwise, let *TXN* be the next SQL-transaction for the SQL-agent.

2) If IMMEDIATE is specified, and if the cursor mode of *TXN* is *cascade off*, then an exception condition is raised: *invalid constraint mode state*.

3) If IMMEDIATE is specified, then

   Case:

   a) If ALL is specified, then the constraint mode in *TXN* of all constraints that are DEFERRABLE is set to *immediate*.

   b) Otherwise, the constraint mode in *TXN* for the constraints identified by the <constraint name>s in the <constraint name list> is set to *immediate*.

4) If DEFERRED is specified, then

   Case:

   a) If ALL is specified, then the constraint mode in *TXN* of all constraints that are DEFERRABLE is set to *deferred*.

   b) Otherwise, the constraint mode in *TXN* for the constraints identified by the <constraint name>s in the <constraint name list> is set to *deferred*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <set constraints mode statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

16.3 <test completion statement>

Function

Test the completion of one or more asynchronous SQL-statements.

Format

```
<test completion statement> ::=
      { TEST | WAIT } { ALL | ANY | <async statement identifier list> } COMPLETION

<async statement identifier list> ::=
      <left paren> <async statement identifier>
          [ { <comma> <async statement identifier> }... ] <right paren>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) If there is no outstanding asynchronous <SQL procedure statement>, then an exception condition is raised: *no outstanding asynchronous SQL statement*.

2) Case:

   a) If WAIT is specified, then

      Case:

      i) If ALL is specified, then the DBMS shall wait until all outstanding asynchronous <SQL procedure statement>s have completed. All such <SQL procedure statement>s are said to have been *successfully tested for completion*.

      ii) If ANY is specified, then the DBMS shall wait until at least one outstanding asynchronous <SQL procedure statement> has completed that has not already been successfully tested for completion. The <SQL procedure statement> that satisfies the WAIT ANY is said to have been *successfully tested for completion*.

      iii) Otherwise, the DBMS shall wait until all asynchronous <SQL procedure statement>s identified by <async statement identifier>s in the <async statement identifier list> have completed. All such <SQL procedure statement>s are said to have been *successfully tested for completion*.

   b) If TEST is specified, then

Case:

i) If ALL is specified and if any asynchronous <SQL procedure statement> is outstanding, then an exception condition is raised: *SQL statement not yet complete*. Any <SQL procedure statement>s that have completed are said to have been *successfully tested for completion*.

ii) If ANY is specified and if no asynchronous <SQL procedure statement> has completed without having been successfully tested for completion, then an exception condition is raised: *SQL statement not yet complete*. Any <SQL procedure statement>s that have completed are said to have been *successfully tested for completion*.

iii) Otherwise, if any of the asynchronous <SQL procedure statement>s identified by <async statement identifier>s in the <async statement identifier list> have not completed, then an exception condition is raised: *SQL statement not yet complete*. Any <SQL procedure statement>s identified by <async statement identifier>s in the <async statement identifier list> that have completed are said to have been *successfully tested for completion*.

3) If any <SQL procedure statement> successfully tested for completion by the <test completion statement> received an SQLSTATE value other than ''00000'', then the SQLSTATE subcode is set to *asynchronous SQL statement returned an SQLSTATE value*.

4) All parameters of the <routine> that contained the asynchronous <SQL procedure statement>, including any status parameters, are set to the values that they would have received had the <SQL procedure statement> been executed synchronously.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <test completion statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

16.4 <savepoint statement>

Function

Establish a savepoint.

Format

```
<savepoint statement> ::= SAVEPOINT <savepoint specifier>
```

Syntax Rules

1) If the <savepoint specifier> is specified as a <simple target specification>, let *T* be the <simple target specification>. The data type of *T* shall be exact numeric with scale 0.

Access Rules

None.

General Rules

1) If <savepoint specifier> is specified as <savepoint name>, then let *S* be the <identifier> specified as <savepoint name>; otherwise, let *S* be the value of *T*.

2) If <savepoint specifier> is specified as <simple target specification> and *S* is not 0 and does not identify an existing savepoint established within the current SQL-transaction, then an exception condition is raised: *savepoint exception—invalid specification*.

3) If *S* identifies an existing savepoint established within the current SQL-transaction, then that savepoint is destroyed.

4) If the number of savepoints that now exist within the current SQL-transaction is equal to the implementation-defined maximum number of savepoints per SQL-transaction, then an exception condition is raised: *exception condition—too many*.

5) If <savepoint specifier> is specified as <simple target specification>, then *S* is set to an implementation-dependent value that is non-0 and different from all other values that have been used to identify savepoints within the current SQL-transaction.

6) A savepoint is established at the current point in the current SQL-transaction and *S* is assigned as the identifier of that savepoint.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <savepoint statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

16.5 <release savepoint statement>

Function

Destroy a savepoint.

Format

```
<release savepoint statement> ::=
       RELEASE SAVEPOINT <savepoint specifier>
```

Syntax Rules

1) If <savepoint specifier> is specified as <simple target specification>, then let *T* be that <simple target specification>. The data type of *T* shall be exact numeric.

Access Rules

None.

General Rules

1) If <savepoint specifier> is specified as <savepoint name>, then let *S* be the identifier specified as <savepoint name>; otherwise, let *S* be the value of *T*.

2) If *S* does not identify a savepoint defined within the current SQL-transaction, then an exception condition is raised: *savepoint exception—invalid specification*.

3) The savepoint identified by *S* and all savepoints established in the current SQL-transaction subsequent to the establishment of S are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <release savepoint statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

16.6 <commit statement>

Function

Terminate the current SQL-transaction with commit.

Format

```
<commit statement> ::=
       COMMIT [ WORK ]
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) If the current SQL-transaction is part of an encompassing transaction that is controlled by an agent other than the SQL-agent, then an exception condition is raised: *invalid transaction termination*.

2) The following <test completion statement> is effectively executed:

   WAIT ALL COMPLETION

3) For every open cursor *CR* in any <module> associated with the current SQL-transaction, the following statement is implicitly executed:

   CLOSE *CR*

4) For every temporary table in any <module> associated with the current SQL-transaction that specifies the ON COMMIT DELETE option and that was updated by the current SQL-transaction, the execution of the <commit statement> is effectively preceded by the execuion of a <delete statement: searched> that specifies DELETE FROM *T*, where *T* is the <table name> of that temporary table. If the cursor mode of the current SQL-transaction is *cascade off*, then the cursor mode of the current SQL-transaction is set to *cascade on*.

5) The effects specified in the General Rules of Subclause 16.2, "<set constraints mode statement>" occur as if the statement SET CONSTRAINTS ALL IMMEDIATE were executed.

6) Any triggered actions that were deferred are effectively executed.

7) Case:

   a) If any constraint is not satisfied, then any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled and an exception condition is raised: *transaction rollback—integrity constraint violation*.

b) If the execution of any <triggered SQL statement> is unsuccessful, then any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled and an exception condition is raised: *transaction rollback—triggered action exception*.

c) If any other error preventing commitment of the SQL-transaction has occurred, then any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled and an exception condition is raised: *transaction rollback* with an implementation-defined subclass value.

d) Otherwise, any changes to SQL-data or schemas that were made by the current SQL-transaction are made accessible to all concurrent and subsequent SQL-transactions.

8) Any savepoints established in the current SQL-transaction are destroyed.

9) The current SQL-transaction is terminated.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) In conforming Entry SQL language, WORK shall be specified.

16.7 <rollback statement>

Function

Terminate the current SQL-transaction with rollback, or rollback all database actions since the establishment of a savepoint.

Format

```
<rollback statement> ::=
       ROLLBACK [ WORK ]
         [ <savepoint clause> ]

<savepoint clause> ::=
       TO SAVEPOINT <savepoint specifier>

<savepoint specifier> ::=
         <savepoint name>
       | <simple target specification>
```

Syntax Rules

1) If <savepoint specifier> is specified as <simple target specification>, then let *T* be that <simple target specification>. The data type of *T* shall be exact numeric.

Access Rules

None.

General Rules

1) If the current SQL-transaction is part of an encompassing transaction that is controlled by an agent other than the SQL-agent and the <rollback statement> is not being implicitly executed, then an exception condition is raised: *invalid transaction termination*.

2) Any outstanding asynchronous <SQL procedure statement> is terminated.

3) If a <savepoint clause> is not specified, then:

   a) Any outstanding asynchronous <SQL procedure statement> is terminated.

   b) Any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled.

   c) All savepoints defined by the current SQL-transaction are destroyed.

   d) Any open cursors in any <module> associated with the current SQL-transaction that were opened by the current SQL-transaction subsequent to the establishment of *S* are closed.

   e) The current SQL-transaction is terminated.

4) If a <savepoint clause> is specified, then:

   a) If <savepoint name> is specified as <simple target specification>, then let *S* be the <identifier> specified as <savepoint name>; otherwise, let *S* be the value of *T*.

   b) If *S* does not specify a savepoint established within the current SQL-transaction, then an exception condition is raised: *savepoint exception—invalid specification*.

   c) Any changes to SQL-data or schemas that were made by the current SQL-transaction subsequent to the establishment of *S* are canceled.

   d) All savepoints established by the current SQL-transaction subsequent to the establishment of *S* are destroyed.

   e) For every open cursor *CR* in any <module> associated with the current SQL-transaction that was opened subsequent to the establishment of *S*, the following statement is implicitly executed:

      CLOSE *CR*

****Editor's Note****
The wording of the preceding rule in light of SQL3 savepoints has been identified as a Possible Problem. See Possible Problem **164** in the Editor's Notes. Furthermore, this rule seems redundant with a rule just above!

   f) The status of any open cursors in any <module> associated with the current SQL-transaction that were opened by the current SQL-transaction before the establishment of *S* is implementation-defined.

      Note: The current SQL-transaction is not terminated, and there is no other effect on the SQL-data or schemas.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <rollback statement> shall contain no <savepoint clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) In conforming Entry SQL language, WORK shall be specified.

17 Connection management

17.1 <connect statement>

Function

Establish an SQL-session.

Format

```
<connect statement> ::=
       CONNECT TO <connection target>

<connection target> ::=
         <SQL-server name>
           [ AS <connection name> ]
           [ USER <user name> ]
       | DEFAULT
```

Syntax Rules

1) If <user name> is not specified, then an implementation-defined <user name> for the SQL-connection is implicit.

Access Rules

None.

General Rules

1) If a <connect statement> is executed after the first transaction-initiating SQL-statement executed by the current SQL-transaction and the implementation does not support transactions that affect more than one SQL-server, then an exception condition is raised: *feature not supported—multiple server transactions*.

2) If <user name> is specified, then let *S* be the character string that is the value of <user name> and let *V* be the character string that is the value of

   TRIM (BOTH ' ' FROM *S*)

3) If *V* does not conform to the Format and Syntax Rules of an <authorization identifier>, then an exception condition is raised: *invalid authorization specification*.

4) If the <module> that contains the <routine> that contains the <connect statement> specifies a <module authorization identifier>, then whether or not <user name> must be identical to that <module authorization identifier> is implementation-defined, as are any other restrictions on the value of <user name>. Otherwise, any restrictions on the value of <user name> are implementation-defined.

5) If the value of <user name> violates the implementation-defined restrictions, then an exception condition is raised: *invalid authorization specification*.

6) If <connection name> was specified, then let *CV* be the value of the <simple value specification> immediately contained in <connection name>. If neither DEFAULT nor <connection name> were specified, then let *CV* be the value of <SQL-server name>. Let *CN* be the result of

   TRIM (BOTH ' ' FROM *CV*)

   If *CN* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid connection name*.

7) If an SQL-connection with name *CN* has already been established by the current SQL-agent and has not been disconnected, or if DEFAULT is specified and a default SQL-connection has already been established by the current SQL-agent and has not been disconnected, then an exception condition is raised: *connection exception—connection name in use*.

8) Case:

   a) If DEFAULT is specified, then the default SQL-session is initiated and associated with the default SQL-server. The method by which the default SQL-server is determined is implementation-defined.

   b) Otherwise, an SQL-session is initiated and associated with the SQL-server identified by <SQL-server name>. The method by which <SQL-server name> is used to determine the appropriate SQL-environment is implementation-defined.

9) If the <connect statement> successfully initiates an SQL-session, then:

   a) The current SQL-connection and current SQL-session, if any, become a dormant SQL-connection and a dormant SQL-session, respectively. The SQL-session context information is preserved and is not affected in any way by operations performed over the initiated SQL-connection.

      Note: The SQL-session context information is defined in Subclause 4.39, "SQL-sessions".

   b) The SQL-session initiated by the <connect statement> becomes the current SQL-session and the SQL-connection established to that SQL-session becomes the current SQL-connection.

   Note: If the <connect statement> fails to initiate an SQL-session, then the current SQL-connection and current SQL-session, if any, remain unchanged.

10) If the SQL-client cannot establish the SQL-connection, then an exception condition is raised: *connection exception— SQL-client unable to establish SQL-connection*.

11) If the SQL-server rejects the establishment of the SQL-connection, then an exception condition is raised: *connection exception— SQL-server rejected establishment of SQL-connection*.

12) The SQL-server for the subsequent execution of <routine>s in any <module>s associated with the SQL-agent is set to the SQL-server identified by <SQL-server name>.

13) The SQL-session <authorization identifier> is set to <user name>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <connect statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

17.2 <set connection statement>

Function

Select an SQL-connection from the available SQL-connections.

Format

```
<set connection statement> ::=
       SET CONNECTION <connection object>

<connection object> ::=
         DEFAULT
       | <connection name>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) If a <set connection statement> is executed after the first transaction-initiating SQL-statement executed by the current SQL-transaction and the implementation does not support transactions that affect more than one SQL-server, then an exception condition is raised: *feature not supported—multiple server transactions*.

2) Case:

   a) If DEFAULT is specified and there is no default SQL-connection that is current or dormant for the current SQL-agent, then an exception condition is raised: *connection exception—connection does not exist*.

   b) Otherwise, if <connection name> does not identify an SQL-session that is current or dormant for the current SQL-agent, then an exception condition is raised: *connection exception—connection does not exist*.

3) If the SQL-connection identified by <connection object> cannot be selected, then an exception condition is raised: *connection exception—connection failure*.

4) The current SQL-connection and current SQL-session become a dormant SQL-connection and a dormant SQL-session, respectively. The SQL-session context information is preserved and is not affected in any way by operations performed over the selected SQL-connection.

   Note: The SQL-session context information is defined in Subclause 4.39, "SQL-sessions".

5) The SQL-connection identified by <connection object> becomes the current SQL-connection and the SQL-session associated with that SQL-connection becomes the currenct SQL-session. All SQL-session context information is restored to the same state as at the time the SQL-connection became dormant.

Note: The SQL-session context information is defined in Subclause 4.39, "SQL-sessions".

6) The SQL-server for the subsequent execution of <routine>s in any <module>s associated with the SQL-agent are set to that of the current SQL-connection.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <set connection statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

17.3 <disconnect statement>

Function

Terminate an SQL-connection.

Format

```
<disconnect statement> ::=
       DISCONNECT <disconnect object>

<disconnect object> ::=
         <connection object>
       | ALL
       | CURRENT
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) If <connection name> is specified and <connection name> does not identify an SQL-connection that is current or dormant for the current SQL-agent, then an exception condition is raised: *connection exception—connection does not exist*.

2) If DEFAULT is specified and there is no default SQL-connection that is current or dormant for the current SQL-agent, then an exception condition is raised: *connection exception—connection does not exist*.

3) If CURRENT is specified and there is no current SQL-connection for the current SQL-agent, then an exception condition is raised: *connection exception—connection does not exist*.

4) Let *C* be the current SQL-connection.

5) Let *L* be a list of SQL-connections. If a <connection name> is specified, then *L* is that SQL-connection. If CURRENT is specified, then *L* is the current SQL-connection. If ALL is specified, then *L* is a list representing every SQL-connection that is current or dormant for the current SQL-agent, in an implementation-dependent order. If DEFAULT is specified, then *L* is the default SQL-connection.

6) If any SQL-connection in *L* is active, then an exception condition is raised: *invalid transaction state*.

7) For every SQL-connection *C1* in *L*, treating the SQL-session *S1* identified by *C1* as the current SQL-session, all of the actions that are required after the last call of a <routine> by an SQL-agent, except for the execution of a <rollback statement> or a <commit statement>, are performed. *C1* is terminated, regardless of any exception condition that might occur during the disconnection process.

Note: See the General Rules of Subclause 13.1, "<module>", for the actions to be performed after the last call of a <routine> by an SQL-agent.

8) If any error is detected during execution of a <disconnect statement>, then a completion condition is raised: *warning—disconnect error*.

9) If *C* is contained in *L*, then there is no current SQL-connection following the execution of the <disconnect statement>. Otherwise, *C* remains the current SQL-connection.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <disconnect statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18 Session management

18.1 <set catalog statement>

Function

Set the default catalog name for unqualified <schema name>s in <preparable statement>s that are prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> and in <direct SQL statement>s that are invoked directly.

Format

```
<set catalog statement> ::=
       SET CATALOG <value specification>
```

Syntax Rules

1) The <data type> of the <value specification> shall be an SQL character data type.

Access Rules

None.

General Rules

1) Let *S* be the character string that is the value of the <value specification> and let *V* be the character string that is the value of

   TRIM (BOTH ' ' FROM *S*)

2) If *V* does not conform to the Format and Syntax Rules of a <catalog name>, then an exception condition is raised: *invalid catalog name*.

3) The default catalog name of the current SQL-session is set to *V*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <set catalog statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2 <set schema statement>

Function

Set the default schema name for unqualified <schema qualified name>s in <preparable statement>s that are prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> and in <direct SQL statement>s that are invoked directly.

Format

```
<set schema statement> ::=
       SET SCHEMA <value specification>
```

Syntax Rules

1) The data type of the <value specification> shall be an SQL character data type.

Access Rules

None.

General Rules

1) Let *S* be the character string that is the value of the <value specification> and let *V* be the character string that is the value of

   TRIM (BOTH ' ' FROM *S*)

2) If *V* does not conform to the Format and Syntax Rules of a <schema name>, then an exception condition is raised: *invalid schema name*.

3) Case:

   a) If *V* conforms to the Format and Syntax Rules for a <schema name> that contains a <catalog name>, then let *X* be the <catalog name> part and let *Y* be the <unqualified schema name> part of *V*. The following statement is implicitly executed:

      SET CATALOG '*X*'

      and the <set schema statement> is effectively replaced by:

      SET SCHEMA '*Y*'

   b) Otherwise, the default unqualified schema name of the current SQL-session is set to *V*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <set schema statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

18.3 <set names statement>

Function

Set the default character set name for <identifier>s and <character string literal>s in <preparable statement>s that are prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> and in <direct SQL statement>s that are invoked directly.

Format

```
<set names statement> ::=
       SET NAMES <value specification>
```

Syntax Rules

1) The <data type> of the <value specification> shall be an SQL character data type.

Access Rules

None.

General Rules

1) Let *S* be the character string that is the value of the <value specification> and let *V* be the character string that is the value of

   TRIM (BOTH ' ' FROM *S*)

2) If *V* does not conform to the Format and Syntax Rules of a <character set name>, then an exception condition is raised: *invalid character set name*.

3) The default character set name of the current SQL-session is set to *V*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <set names statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.4 <set session authorization identifier statement>

Function

Set the <authorization identifier> of the current SQL-session.

Format

```
<set session authorization identifier statement> ::=
       SET SESSION AUTHORIZATION <value specification>
```

Syntax Rules

1) The <data type> of the <value specification> shall be an SQL character data type.

Access Rules

None.

General Rules

1) If a <set session authorization identifier statement> is executed and an SQL-transaction is currently active, then an exception condition is raised: *invalid transaction state*.

2) Let *S* be the character string that is the value of the <value specification> and let *V* be the character string that is the value of

   TRIM (BOTH ' 'FROM *S*)

3) If *V* does not conform to the Format and Syntax Rules of an <authorization identifier>, then an exception condition is raised: *invalid authorization specification*.

4) Whether or not the <authorization identifier> for the SQL-session can be set to an <authorization identifier> other than the <authorization identifier> of the SQL-session when the SQL-session is started is implementation-defined, as are any restrictions pertaining to such changes.

5) If the current <authorization identifier> is restricted from setting the <authorization identifier> to the specified value, then an exception condition is raised: *invalid authorization specification*.

6) Let *T* be any temporary table or temporary view defined in the currently active SQL-session. In all the privilege descriptors for *T* and for each of the columns of *T*, the <authorization identifier> is set to *V*.

7) The <authorization identifier> of the current SQL-session is set to *V*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any <set session authorization identifier statement>.

18.5 <set role statement>

Function

Set enabled applicable roles for the current SQL-session.

Format

```
<set role statement> ::= SET ROLE <role name>
```

Syntax Rules

None.

Access Rules

1) A role authorization descriptor shall exist that indicates that the role identified by <role name> has been granted to the <authorization identifier> of the current SQL-session or to PUBLIC.

General Rules

1) If a <set role statement> is executed and an SQL-transaction is currently active, then an exception condition is raised: *invalid transaction state*.

2) Let *R* be the set of roles defined by the specified <role name> and all other roles contained in that role.

3) The set of applicable privileges for the <authorization identifier> of the current SQL-session is set to contain those privileges defined by privilege descriptors associated with the roles in *R* or defined by privilege descriptors associated with that <authorization identifier>, together with those defined by privilege descriptors associated with PUBLIC.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <set role statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.6 <set local time zone statement>

Function

Set the default local time zone displacement for the current SQL-session.

Format

```
<set local time zone statement> ::=
       SET TIME ZONE <set time zone value>

<set time zone value> ::=
         <interval value expression>
       | LOCAL
```

Syntax Rules

1) The <data type> of the <interval value expression> immediately contained in the <set time zone value> shall be INTERVAL HOUR TO MINUTE.

Access Rules

None.

General Rules

1) Case:

   a) If LOCAL is specified, then the default local time zone displacement of the current SQL-session is set to the original implementation-defined default local time zone displacement that was established when the current SQL-session was started.

   b) Otherwise,

      Case:

      i) If the value of the <interval value expression> is not the null value and is between INTERVAL −'12:59' and INTERVAL +'13:00', then the default local time zone displacement of the current SQL-session is set to the value of the <interval value expression>.

      ii) Otherwise, an exception condition is raised: *data exception—invalid time zone displacement value*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restsrictions:

   a) Conforming Entry SQL language shall not contain any <set local time zone statement>.

19 Dynamic SQL

****Editor's Note****
The fact that prepared statements can be shared among modules, but cursors cannot has been identified as a Possible Problem. See Possible Problem **179** in the Editor's Notes.

19.1 Description of SQL item descriptor areas

Function

Specify the identifiers, data types, and codes used in SQL item descriptor areas.

Syntax Rules

1) An SQL item descriptor area comprises the items specified in Table 24, "Data types of <key word>s used in SQL item descriptor areas".

2) Let *DT* be a data type. The data type *T* of a <simple value specification> or a <simple target specification> *SVT* is said to *match* the data type specified by the item descriptor area if and only if one of the following conditions is true.

   Case:

   a) TYPE indicates NUMERIC and *T* is specified by NUMERIC(*P*,*S*), where *P* is the value of PRECISION and *S* is the value of SCALE.

   b) TYPE indicates DECIMAL and *T* is specified by DECIMAL(*P*,*S*), where *P* is the value of PRECISION and *S* is the value of SCALE.

   c) TYPE indicates INTEGER and *T* is specified by INTEGER.

   d) TYPE indicates SMALLINT and *T* is specified by SMALLINT.

   e) TYPE indicates FLOAT and *T* is specified by FLOAT(*P*), where *P* is the value of PRECISION.

   f) TYPE indicates REAL and *T* is specified by REAL.

   g) TYPE indicates DOUBLE PRECISION and *T* is specified by DOUBLE PRECISION.

   h) TYPE indicates BIT and *T* is specified by BIT(*L*), where *L* is the value of LENGTH.

   i) TYPE indicates BIT VARYING and *T* is specified by BIT VARYING(*L*), where

      Case:

      i) *SVT* is a <simple value specification> and *L* is the value of LENGTH.

      ii) *SVT* is a <simple target specification> and *L* is not less than the value of LENGTH.

j) TYPE indicates CHARACTER and *T* is specified by CHARACTER(*L*), where *L* is the value of LENGTH and the <character set specification> formed by the values of CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME identifies the character set of *SVT*.

k) TYPE indicates CHARACTER VARYING and *T* is specified by CHARACTER VARYING(*L*), where the <character set specification> formed by the values of CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME identifies the character set of *SVT* and

Case:

i) *SVT* is a <simple value specification> and *L* is the value of LENGTH.

ii) *SVT* is a <simple target specification> and *L* is not less than the value of LENGTH.

3) An item descriptor area is *valid* if and only if TYPE indicates a code defined in Table 25, "Codes used for SQL data types in Dynamic SQL", and one of the following is true:

Case:

a) TYPE indicates NUMERIC and PRECISION and SCALE are valid precision and scale values for the NUMERIC data type.

b) TYPE indicates DECIMAL and PRECISION and SCALE are valid precision and scale values for the DECIMAL data type.

c) TYPE indicates FLOAT and PRECISION is a valid precision value for the FLOAT data type.

d) TYPE indicates INTEGER, SMALLINT, REAL, or DOUBLE PRECISION.

e) TYPE indicates BIT or BIT VARYING and LENGTH is a valid length value for the BIT date type.

f) TYPE indicates CHARACTER or CHARACTER VARYING, LENGTH is a valid length value for the CHARACTER data type, and CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are a valid qualified character set name for the CHARACTER data type.

g) TYPE indicates a <datetime type>, DATETIME_INTERVAL_CODE is a code specified in Table 26, "Codes associated with datetime data types in Dynamic SQL", and PRECISION is a valid value for the <time precision> or <timestamp precision> of the indicated datetime datatype.

h) TYPE indicates an <interval type>, DATETIME_INTERVAL_CODE is a code specified in Table 27, "Codes used for <interval qualifier>s in Dynamic SQL", and DATETIME_INTERVAL_PRECISION and PRECISION are valid values for <interval leading field precision> and <interval fractional seconds precision> for an <interval qualifier>.

Table 24—Data types of <key word>s used in SQL item descriptor areas

<key word>	Data Type
TYPE	exact numeric with scale 0
LENGTH	exact numeric with scale 0
OCTET_LENGTH	exact numeric with scale 0
RETURNED_LENGTH	exact numeric with scale 0
RETURNED_OCTET_LENGTH	exact numeric with scale 0
PRECISION	exact numeric with scale 0
SCALE	exact numeric with scale 0
DATETIME_INTERVAL_CODE	exact numeric with scale 0
DATETIME_INTERVAL_ PRECISION	exact numeric with scale 0
NULLABLE	exact numeric with scale 0
KEY_MEMBER	exact numeric with scale 0
INDICATOR	exact numeric with scale 0
DATA	matches data type specified by TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_ PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_ SCHEMA, and CHARACTER_SET_NAME
NAME	character string with character set SQL_TEXT and length not less than 128 characters
UNNAMED	exact numeric with scale 0
COLLATION_CATALOG	character string with character set SQL_TEXT and length not less than 128 characters
COLLATION_SCHEMA	character string with character set SQL_TEXT and length not less than 128 characters
COLLATION_NAME	character string with character set SQL_TEXT and length not less than 128 characters
CHARACTER_SET_CATALOG	character string with character set SQL_TEXT and length not less than 128 characters
CHARACTER_SET_SCHEMA	character string with character set SQL_TEXT and length not less than 128 characters
CHARACTER_SET_NAME	character string with character set SQL_TEXT and length not less than 128 characters
NULL_CLASS_CATALOG	character string with character set SQL_TEXT and length not less than 128 characters
NULL_CLASS_SCHEMA	character string with character set SQL_TEXT and length not less than 128 characters
NULL_CLASS_NAME	character string with character set SQL_TEXT and length not less than 128 characters

Access Rules

None.

General Rules

1) Table 25, "Codes used for SQL data types in Dynamic SQL", specifies the codes associated with the SQL data types.

Table 25—Codes used for SQL data types in Dynamic SQL

Data Type	Code
Implementation-defined data types	< 0
Abstract data types	17
BIT	14
BIT VARYING	15
BOOLEAN	16
CHARACTER	1
CHARACTER VARYING	12
DATE, TIME, or TIMESTAMP	9
DECIMAL	3
DOUBLE PRECISION	8
ENUMERATED	13
FLOAT	6
INTEGER	4
INTERVAL	10
NUMERIC	2
REAL	7
SMALLINT	5

2) Table 26, "Codes associated with datetime data types in Dynamic SQL", specifies the codes associated with the datetime data types.

Table 26—Codes associated with datetime data types in Dynamic SQL

Code	Datetime Data Type
1	DATE
2	TIME
3	TIMESTAMP
4	TIME WITH TIME ZONE
5	TIMESTAMP WITH TIME ZONE

3) Table 27, "Codes used for <interval qualifier>s in Dynamic SQL", specifies the codes associated with <interval qualifier>s for interval data types.

Table 27—Codes used for <interval qualifier>s in Dynamic SQL

Code	Datetime Qualifier
1	YEAR
2	MONTH
3	DAY
4	HOUR
5	MINUTE
6	SECOND
7	YEAR TO MONTH
8	DAY TO HOUR
9	DAY TO MINUTE
10	DAY TO SECOND
11	HOUR TO MINUTE
12	HOUR TO SECOND
13	MINUTE TO SECOND

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

19.2 <allocate descriptor statement>

Function

Allocate an SQL descriptor area.

Format

```
<allocate descriptor statement> ::=
       ALLOCATE DESCRIPTOR <descriptor name> [ WITH MAX <occurrences> ]

<occurrences> ::= <simple value specification>
```

Syntax Rules

1) The data type of <occurrences> shall be exact numeric with scale 0.

2) If WITH MAX <occurrences> is not specified, then an implementation-defined default value for <occurrences> that is greater than 0 is implicit.

Access Rules

None.

General Rules

1) Let *S* be the character string that is the value of the <simple value specification> that is immediately contained in <descriptor name> and let *V* be the character string that is the result of

   TRIM (BOTH ' ' FROM *S*)

   If *V* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid SQL descriptor name*.

2) The <allocate descriptor statement> allocates an SQL descriptor area whose name is *V* and whose scope is specified by the <scope clause>. The descriptor area will have at least <occurrences> number of item descriptor areas. All values are initially undefined. If an SQL descriptor has already been allocated whose name is *V*, whose scope is specified by the <scope option>, and that has not yet been deallocated, then an exception condition is raised: *invalid SQL descriptor name*.

3) If <occurrences> is less than 1 or is greater than an implementation-defined maximum value, then an exception condition is raised: *dynamic SQL error—invalid descriptor index*. The maximum number of SQL descriptor areas that can be allocated at one time is implementation-defined.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) An <occurrences> and a <descriptor name> shall be a <literal>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.3 <deallocate descriptor statement>

Function

Deallocate an SQL descriptor area.

Format

```
<deallocate descriptor statement> ::=
       DEALLOCATE DESCRIPTOR <descriptor name>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) The <deallocate descriptor statement> deallocates an SQL descriptor area whose name is the value of the <descriptor name>'s <simple value specification> and whose scope is specified by the <scope clause>. If an SQL descriptor is not currently allocated whose name is the value of the <descriptor name>'s <simple value specification> and whose scope is specified by the <scope option>, then an exception condition is raised: *invalid SQL descriptor name*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <descriptor name> shall be a <literal>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.4 <get descriptor statement>

Function

Get information from an SQL descriptor area.

Format

```
<get descriptor statement> ::=
       GET DESCRIPTOR <descriptor name> <get descriptor information>

<get descriptor information> ::=
         <get count>
       | <get key type>
       | VALUE <item number>
           <get item information> [ { <comma> <get item information> }... ]

<get count> ::=
       <simple target specification 1> <equals operator> COUNT

<get key type> ::=
       <simple target specification 2> <equals operator> KEY_TYPE

<get item information> ::=
       <simple target specification 3> <equals operator> <descriptor item name>

<item number> ::= <simple value specification>

<simple target specification 1> ::= <simple target specification>

<simple target specification 2> ::= <simple target specification>

<simple target specification 3> ::= <simple target specification>

<descriptor item name> ::=
         TYPE
       | LENGTH
       | OCTET_LENGTH
       | RETURNED_LENGTH
       | RETURNED_OCTET_LENGTH
       | PRECISION
       | SCALE
       | DATETIME_INTERVAL_CODE
       | DATETIME_INTERVAL_PRECISION
       | NULLABLE
       | KEY_MEMBER
       | INDICATOR
       | DATA
       | NAME
       | UNNAMED
       | COLLATION_CATALOG
       | COLLATION_SCHEMA
       | COLLATION_NAME
       | CHARACTER_SET_CATALOG
       | CHARACTER_SET_SCHEMA
       | CHARACTER_SET_NAME
```

Syntax Rules

1) The data type of <item number> shall be exact numeric with scale 0.

2) The data type of <simple target specification 1> and of <simple target specification 2> shall be exact numeric with scale 0.

3) The data type of a <simple target specification 3> shall be the data type shown in Table 24, "Data types of <key word>s used in SQL item descriptor areas", for the corresponding <descriptor item name>.

Access Rules

None.

General Rules

1) If a <descriptor name> specified in a <get descriptor statement> identifies an SQL descriptor area that is not currently allocated, then an exception condition is raised: *invalid SQL descriptor name*.

2) If the data type of the <simple target specification> associated with the keyword DATA does not match the data type specified by TYPE, LENGTH, the item descriptor area, then an exception condition is raised: *data exception—error in assignment*.

   Note: ''Match'' is defined in the Syntax Rules of Subclause 19.1, "Description of SQL item descriptor areas".

3) If the <item number> specified in a <get descriptor statement> is greater than the number of <occurrences> specified when the <descriptor name> was allocated or less than 1, then an exception condition is raised: *dynamic SQL error—invalid descriptor index*.

4) If the <item number> specified in a <get descriptor statement> is greater than the value of COUNT, then a completion condition is raised: *no data*.

5) If a <get descriptor statement> is executed to get the value of DATA without getting the value of INDICATOR and the value of INDICATOR is negative, then an exception condition is raised: *data exception—null value, no indicator parameter*.

6) If an exception condition is raised in a <get descriptor statement>, then the values of all targets specified by <simple target specification 1>, <simple target specification 2>, and <simple target specification 3> are implementation-dependent.

7) A <get descriptor statement> gets values in the SQL descriptor area specified by <descriptor name>. The values are as follows:

   a) COUNT is a count of the number of <dynamic parameter specification>s or <select list> columns described in item descriptor areas.

   b) Case:

      a) If the SQL item descriptor area identified by <descriptor name> is a collection *SL* of <select list> columns of some table *T* and some subset of *SL* is the primary key of *T*, then KEY_TYPE is 1.

b) If the SQL item descriptor area identified by <descriptor name> is a collection *SL* of <select list> columns of some table *T* and some subset of *SL* is the preferred key of *T*, then KEY_TYPE is 2.

Note: Primary keys and preferred keys of tables are defined in Subclause 4.18, "Functional dependencies and candidate keys".

c) Otherwise, KEY_TYPE is 0.

c) The values of TYPE, LENGTH, OCTET_LENGTH, RETURNED_LENGTH, RETURNED_OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are the data type, length (in characters, bits, or positions, as appropriate), length in octets, returned length in characters, returned length in octets, precision, scale, datetime data type, interval qualifier, collation, and character set of a <dynamic parameter specification> or <select list> column described by the item descriptor area specified by <item number>.

d) NAME is the name associated with the <dynamic parameter specification> or <select list> column described by the item descriptor area specified by <item number>. For a <select list> column, if the column name is implementation-dependent, then NAME is the implementation-dependent name for the column and UNNAMED is set to 1; otherwise, NAME is the <derived column name> of the column and UNNAMED is set to 0. For a <dynamic parameter specification>, the values of NAME and UNNAMED are implementation-dependent.

e) DATA is a value for the <target specification> described by the item descriptor area specified by <item number>. If the value of INDICATOR is negative, then the value of DATA is undefined.

f) INDICATOR is a value for the <indicator parameter> associated with the <target specification>.

g) Case:

i) For a <select list> column, if NULLABLE is set to 1, then the column can have a null value; otherwise, the column cannot have a null value.

ii) For a <dynamic parameter specification>, NULLABLE is set to 1, indicating that the <dynamic parameter specification> can have the null value.

h) Case:

a) For a <select list> column *C*, KEY_MEMBER is 1 if and only if *C* is a member of the primary key or preferred key of *T*.

b) Otherwise, KEY_MEMBER is 0.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <descriptor name> shall be a <literal>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.5 <set descriptor statement>

Function

Set information in an SQL descriptor area.

Format

```
<set descriptor statement> ::=
       SET DESCRIPTOR <descriptor name> <set descriptor information>

<set descriptor information> ::=
         <set count>
       | VALUE <item number>
           <set item information> [ { <comma> <set item information> }... ]

<set count> ::=
       COUNT <equals operator> <simple value specification 1>

<set item information> ::=
       <descriptor item name> <equals operator> <simple value specification 2>

<simple value specification 1> ::= <simple value specification>

<simple value specification 2> ::= <simple value specification>

<item number> ::= <simple value specification>
```

Syntax Rules

1) The data type of <item number> shall be exact numeric with scale 0.

2) The data type of <simple value specification 1> shall be an exact numeric with scale 0.

3) The data type of a <simple value specification 2> shall be the data type shown in Table 24, "Data types of <key word>s used in SQL item descriptor areas", for the corresponding <descriptor item name>. <descriptor item name> shall not be RETURNED_LENGTH, RETURNED_OCTET_LENGTH, OCTET_LENGTH, NULLABLE, KEY_MEMBER, COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, NAME, or UNNAMED. Other alternatives for <descriptor item name> shall not be specified more than once in a <set descriptor statement>.

Access Rules

None.

General Rules

1) If a <descriptor name> specified in a <set descriptor statement> identifies an SQL descriptor area that is not currently allocated, then an exception condition is raised: *invalid SQL descriptor name*.

2) If the <item number> specified in a <set descriptor statement> is greater than the number of <occurrences> specified when the <descriptor name> was allocated or less than 1, then an exception condition is raised: *dynamic SQL error—invalid descriptor index*.

3) A <set descriptor statement> sets values in the SQL descriptor area specified by <descriptor name>. The values are as follows:

   a) COUNT is a count of the number of <dynamic parameter specification> values or <target specification>s described in item descriptor areas.

   b) TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are a description of the data type, length (in characters, bits, or positions, as appropriate), precision, scale, datetime data type, interval qualifier, and character set of a <dynamic parameter specification> value or <target specification> described by the item descriptor area specified by <item number>.

   c) DATA is a value for the <dynamic parameter specification> described by the item descriptor area specified by <item number>. INDICATOR is a value for the <indicator parameter> associated with the <dynamic parameter specification>.

4) When more than one value is set in a single <set descriptor statement>, the values are effectively assigned in the following order: TYPE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, PRECISION, SCALE, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, LENGTH, INDICATOR, and DATA.

   When any value other than DATA is set, the value of DATA becomes undefined.

5) For every <set item information> specified, let *DIN* be the <descriptor item name> and let *V* be the value of the <simple value specification 2>. If *DIN* is DATA and the data type of *V* does not match the data type specified by the item descriptor area, then an exception condition is raised: *data exception—error in assignment*.

   Note: ''Match'' is defined in the Syntax Rules of Subclause 19.1, "Description of SQL item descriptor areas".

   The item descriptor area field *DIN* is set to *V* and then

   Case:

   a) If *DIN* is TYPE and *V* indicates CHARACTER or CHARACTER VARYING, then CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are set to the values for the default character set name for the SQL-session, LENGTH is set to 1, and all other item descriptor area fields are set to implementation-dependent values.

   b) If *DIN* is TYPE and *V* indicates BIT or BIT VARYING, then LENGTH is set to 1 and all other item descriptor area fields are set to implementation-dependent values.

   c) If *DIN* is TYPE and *V* indicates DATETIME, then PRECISION is set to 0 and all other item descriptor area fields are set to implementation-dependent values.

   d) If *DIN* is TYPE and *V* indicates INTERVAL, then DATETIME_INTERVAL_PRECISION is set to 2 and all other item descriptor area fields are set to implementation-dependent values.

   e) If *DIN* is TYPE and *V* indicates NUMERIC or DECIMAL, then SCALE is set to 0, PRECISION is set to the implementation-defined default value for the precision of NUMERIC or DECIMAL data types, respectively, and all other item descriptor area fields are set to implementation-dependent values.

   f) If *DIN* is TYPE and *V* indicates INTEGER or SMALLINT, then all other item descriptor area fields are set to implementation-dependent values.

g) If *DIN* is TYPE and *V* indicates FLOAT, then PRECISION is set to the implementation-defined default value for the precision of FLOAT data types and all other item descriptor area fields are set to implementation-dependent values.

h) If *DIN* is TYPE and *V* indicates REAL or DOUBLE PRECISION, then all other item descriptor area fields are set to implementation-dependent values.

i) If *DIN* is DATETIME_INTERVAL_CODE and TYPE is DATETIME, then

i) If *V* indicates DATE, TIME, or TIME WITHTIME ZONE, then PRECISION is set to 0 and all other item descriptor area fields are set to implementation-dependent values.

ii) If *V* indicates TIMESTAMP, or TIMESTAMP WITHTIME ZONE, then PRECISION is set to 6 and all other item descriptor area fields are set to implementation-dependent values.

j) If *DIN* is DATETIME_INTERVAL_CODE and TYPE is INTERVAL, then DATETIME_INTERVAL_PRECISION is set to 2 and

i) If *V* indicates DAY TO SECOND, HOUR TO SECOND, MINUTE TO SECOND, or SECOND, then PRECISION is set to 6.

ii) Otherwise, PRECISION is set to 0.

6) If an exception condition is raised in a <set descriptor statement>, then the values of all elements of the item descriptor area specified in the <set descriptor statement> are implementation-dependent.

7) Restrictions on changing TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, and NULL_CLASS_NAME values resulting from the execution of a <describe statement> before execution of a <execute statement>, <dynamic open statement>, or <dynamic fetch statement> are implementation-defined, except as specified in the General Rules of Subclause 19.9, "<using clause>".

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) A <descriptor name> shall be a <literal>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.6 <prepare statement>

Function

Prepare a statement for execution.

Format

```
<prepare statement> ::=
       PREPARE <SQL statement name> FROM <SQL statement variable>

<SQL statement variable> ::= <simple value specification>

<preparable statement> ::=
         <preparable SQL data statement>
       | <preparable SQL schema statement>
       | <preparable SQL transaction statement>
       | <preparable SQL session statement>
       | <call statement>
       | <preparable implementation-defined statement>

<preparable SQL data statement> ::=
         <delete statement: searched>
       | <dynamic single row select statement>
       | <insert statement>
       | <dynamic select statement>
       | <update statement: searched>
       | <preparable dynamic delete statement: positioned>
       | <preparable dynamic update statement: positioned>

<preparable SQL schema statement> ::=
         <SQL schema statement>

<preparable SQL transaction statement> ::=
         <SQL transaction statement>

<preparable SQL session statement> ::=
         <SQL session statement>

<dynamic select statement> ::= <cursor specification>

<dynamic single row select statement> ::= <query specification>

<preparable implementation-defined statement> ::= !! See the Syntax Rules.
```

Syntax Rules

1) The <simple value specification> of <SQL statement variable> shall not be a <literal>.

2) The data type of <SQL statement variable> shall be character string.

3) The Format and Syntax Rules for <preparable implementation-defined statement> are implementation-defined.

Access Rules

None.

General Rules

1) Let *P* be the contents of the <SQL statement variable>.

2) If *P* is a <preparable dynamic delete statement: positioned> or a <preparable dynamic update statement: positioned>, then *P* refers to either a dynamic cursor with the same <cursor name> or to an extended dynamic cursor whose <extended cursor name> value is the same as the <cursor name>.

   Case:

   a) If both an extended dynamic cursor and a dynamic cursor with the same name as the <cursor name> exist, then an exception condition is raised: *ambiguous cursor name*.

   b) If there is neither an extended dynamic cursor nor a dynamic cursor with the name of <cursor name>, then an exception condition is raised: *invalid cursor name*.

3) Let *E1*, *E2*, *E3*, and *E4* be <value expression>s of the form ''<dynamic parameter specification>'' or a <dynamic parameter specification> enclosed in any number of matching parentheses. Let *F* be a <value expression> whose data type is determined via General Rules in Clause 6, "Scalar expressions" and the Rules in this Subclause. Let *X1*, *X2*, *X3*, and *X4* be <value expression>s meeting the criteria for either *En* or *F*.

4) If one or more of the following are true, then an exception condition is raised: *syntax error or access rule violation in dynamic SQL statement*.

   a) *P* does not conform to the Format, Syntax Rules, and Access Rules of a <preparable statement>.

   b) *P* contains *E1* as a <value expression> simply contained in a <select list>.

   c) *P* contains *E1* as both operands of a single dyadic operator.

   d) *P* contains a <value expression> of the form '' + *E1*'', ''– *E1*'', ''*E1* COLLATE <collation name>'', or ''EXTRACT (<extract field> FROM *E1*)''.

   e) *P* contains a <set function specification> with argument *E1*.

   f) *P* contains a <null predicate> with a value *E1* in the <row value constructor>.

   g) *P* contains an <overlaps predicate> where the <value expression> that is the second <row value constructor element> in either the first or second operands of the <overlaps predicate> is *E1*.

   h) *P* contains *E1* as the first operand of COALESCE or the first <when operand> in a <case specification> or both operands of NULLIF.

   i) *P* contains a <comparison predicate> or <between predicate> where both the *i*-th <value expression> in the <row value constructor> that is the first operand and the *i*-th <value expression> in the <row value constructor> that is the second operand of the <comparison predicate> or the second or third operand of the <between predicate> are *E1*.

j) *P* contains a <table value constructor> in which the *i*-th <value expression> in each <row value constructor> is *E1* and either:

i) *P* is not an <insert statement>, or

ii) *P* is an <insert statement> and the <table value constructor> is not the <query expression> simply contained in the <insert statement>.

k) *P* contains an <in predicate> with an <in value list> in which both the <row value constructor> and the first <value specification> of the <in value list> are *E1*.

l) *P* contains a <position expression> in which both immediately contained <character value expression>s are *E1*.

m) *P* contains a <form-of-use conversion> or a <character translation> whose immediately contained <character value expression> is *E1*.

n) *P* contains a <fold> whose operand is *E1*.

o) *P* contains a <trim function> whose <trim character> or <trim source> is *E1*.

p) *P* contains a <character substring function> whose <character value expression> is *E1*.

q) *P* contains a <simple comment>.

r) *P* contains an <argument> that is *E1*.

5) Case:

a) If *E1* is followed by an <interval qualifier> *IQ*, then the data type of *E1* is assumed to be INTERVAL *IQ*.

b) In OCTET_LENGTH(*E1*), CHAR_LENGTH(*E1*), and CHARACTER_LENGTH(*E1*), the data type of *E1* is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING.

c) In POSITION(*X1* IN *X2*), and SUBSTRING(*X1* FROM *X3* FOR *X4*), if *X1* (*X2*) meets the criteria for *E1*, *E2*, *E3*, and *E4*, then the data type of *X1* (*X2*) is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING. If *X3* (*X4*) meets the criteria for *E1*, *E2*, *E3*, and *E4*, then the data type of *X3* (*X4*) is assumed to be NUMERIC(*P*,0), where *P* is the implementation-defined maximum value of <precision> for NUMERIC.

d) In a <value expression> of the form ''*X1* <concatenation operator> *X2*'', if *X1* (*X2*) meets the criteria for *E1*, *E2*, *E3*, and *E4*, then the data type of *X1* (*X2*) is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING.

e) In BIT_LENGTH(*E1*), the data type of *E1* is assumed to be BIT VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for BIT VARYING.

f) In a <value expression> of the form ''*E1* + <datetime term>'', ''<datetime term> + *E1*'' or ''<datetime term> − *E1*'', the data type of *E1* is assumed to be

Case:

i) If the <datetime term> is a date data type, then the data type of *E1* is assumed to be INTERVAL YEAR(*P*) TO MONTH, where *P* is the implementation-defined maximum <interval leading field precision>.

ii) Otherwise, the data type of *E1* is assumed to be INTERVAL DAY(*P*) TO SECOND(*F*), where *P* and *F* are the implementation-defined maximum <interval leading field precision> and maximum <interval fractional seconds precision>, respectively.

g) In a <value expression> of the form ''<interval term> * *E1*'' or ''<interval term> / *E1*'', the data type of *E1* is assumed to be NUMERIC(*P*,0), where *P* is the implementation-defined maximum value of <precision> for NUMERIC.

h) In all other <value expression>s of the form ''*E1* + *F*'', ''*E1* – *F*'', ''*E1* * *F*'', ''*E1* / *F*'', ''*F* + *E1*'', ''*F* – *E1*'', ''*F* * *E1*'', or ''*F* / *E1*'', the data type of *E1* is assumed to be the data type of *F*.

i) In a <value expression> of the form ''CAST (*E1* AS <domain name>)'', ''CAST (*E1* AS <data type>)'', the data type of *E1* is the <data type> of the specified domain or the explicitly-specified <data type>.

j) If one or more operands of COALESCE are *E1*, then the data type of *E1* is assumed to be the data type of the first operand.

k) If one or more <when operand>s in a <case specification> are *E1*, then the data type of *E1* is assumed to be the data type of the first <when operand>.

l) If one operand of NULLIF is *E1*, then the data type of *E1* is assumed to be the data type of the other operand.

m) In the first and second operands of a <comparison predicate> or <between predicate>, or the first and third operands of a <between predicate>, if the *i*-th value of one operand is *E1*, then the data type of E1 is the data type of the *i*-th value of the other operand.

n) In the first and second operands of an <overlaps predicate>, if either of the first <row value constructor element>s is *E1*, then the data type of *E1* is the data type of the first <row value constructor element> of the other operand. If both of the first <row value constructor element>s are *E1*, then the data type of each *E1* is assumed to be TIMESTAMP WITHTIME ZONE.

o) In a <table value constructor> in which the *i*-th <value expression> of some <row value constructor> is *E1* that contains a <row value constructor> whose *i*-th <value expression> is not *E1*, the data type of *E1* is the data type of the *i*-th <value expression> of the first <row value constructor> whose *i*-th <value expression> is not *E1*.

p) In a <table value constructor> in which the *i*-th <value expression> in each <row value constructor> is *E1* that is the <query expression> simply contained in an <insert statement>, the data type of *E1* is the data type of the corresponding column of the implicit or explicit <insert column list> contained in the <insert statement>.

q) In an <in predicate> that specifies a <table subquery>, the data types of <value expression>s *E1* in the <row value constructor> are assumed to be the same as the data types of the respective columns of the <table subquery>.

r) In an <in predicate> that specifies an <in value list>, if the <row value constructor> is not *E1*, then let *D* be its data type. Otherwise, let *D* be the data type of the first <value specification> of the <in value list>. The data type of any *E1* in the <in predicate> is assumed to be *D*.

s) If *E1* appears for <match value>, <pattern>, or <escape character> in <like predicate>, then the data type of *E1* is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING.

t) If any value in the <row value constructor> of a <quantified comparison predicate> or <match predicate> is *E1*, then the data type of *E1* is assumed to be the same as the data type of the respective column of the <table subquery>.

u) If <cycle mark value> is *E1* in <recursive union>, then the data type of *E1* is assumed to be character string of length 1. If <limit expression> contains a <value specification> *E1*, then the data type of *E1* is assumed to be exact numeric with scale 0.

v) If <value specification> in <set catalog statement>, <set schema statement>, <set names statement>, or <set session authorization identifier statement> is *E1*, then the data type of *E1* is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING.

w) If <simple value specification> in <set local time zone statement> is *E1*, then the data type of *E1* is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING.

x) If a <value expression> in a <set clause> is *E1*, then the data type of *E1* is assumed to be the same data type as the corresponding <object column>.

• 1 Rule deleted.

6) If the value of the <SQL statement name> identifies an existing prepared statement, then an implicit

DEALLOCATE PREPARE *SSN*

is executed, where *SSN* is the value of the <SQL statement name>.

7) *P* is prepared for execution.

8) If <extended statement name> is specified for the <SQL statement name>, then let *S* be the character string that is the value of the <simple target specification> and let *V* be the character string that is the result of

TRIM (BOTH ' ' FROM *S*)

If *V* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid SQL statement identifier*.

9) Case:

a) If <extended statement name> is specified for the <SQL statement name>, then the value of the <extended statement name> is associated with the prepared statement. This value and explicit or implied <scope option> shall be specified for each <execute statement> or <allocate cursor statement> that is to be associated with this prepared statement.

b) If <statement name> is specified for the <SQL statement name>, then:

i) If *P* is a <cursor specification> and <statement name> is associated with a cursor *C* through a <dynamic declare cursor>, then an association is made between *C* and *P*. The association is preserved until the prepared statement is destroyed.

ii) If *P* is not a <cursor specification> and <statement name> is associated with a cursor *C* through a <dynamic declare cursor>, then an exception condition is raised: *dynamic SQL error— prepared statement is not a cursor specification*.

iii) Otherwise, the same <statement name> shall be specified for each <execute statement> that is to be associated with this prepared statement.

10) The validity of an <extended statement name> value or a <statement name> in an SQL-transaction different from the one in which the statement was prepared is implementation-dependent.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.7 <deallocate prepared statement>

Function

Deallocate SQL-statements that have been prepared with a <prepare statement>.

Format

```
<deallocate prepared statement> ::=
       DEALLOCATE PREPARE <SQL statement name>
```

Syntax Rules

1) If <SQL statement name> is a <statement name>, then the <module> that contains the <deallocate prepared statement> shall also contain a <prepare statement> that specifies the same <statement name>.

Access Rules

None.

General Rules

1) If the <SQL statement name> does not identify a statement prepared in the scope of the <SQL statement name>, then an exception condition is raised: *invalid SQL statement name*.

2) If the value of <SQL statement name> identifies an existing prepared statement that is the <cursor specification> of an open cursor, then an exception condition is raised: *invalid cursor state*.

3) The prepared statement identified by the <SQL statement name> is destroyed. Any cursor that was allocated with an <allocate cursor statement> that is associated with the prepared statement identified by the <SQL statement name> is destroyed. If the value of the <SQL statement name> identifies an existing prepared statement that is a <cursor specification>, then any prepared statements that reference that cursor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <deallocate prepared statement>.

3) The following restrictions apply for Entry SQL in addition to Intermediate SQL restrictions:

   None.

19.8 <describe statement>

Function

Obtain information about the <select list> columns or <dynamic parameter specification>s contained in a prepared statement.

Format

```
<describe statement> ::=
         <describe input statement>
       | <describe output statement>

<describe input statement> ::=
       DESCRIBE INPUT <SQL statement name> <using descriptor>

<describe output statement> ::=
       DESCRIBE [ OUTPUT ] <SQL statement name> <using descriptor>
```

Syntax Rules

1) If <SQL statement name> is a <statement name>, then the containing <module> shall contain a <prepare statement> whose <statement name> is the same as the <statement name> of the <describe statement>.

Access Rules

None.

General Rules

1) When the <describe statement> is executed, if the value of the <SQL statement name> does not identify a statement prepared in the scope of the <SQL statement name>, then an exception condition is *invalid SQL statement name*.

2) The descriptor area is set with information as described in Subclause 19.9, "<using clause>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <describe input statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.9 <using clause>

Function

Describe the input/output variables for an <SQL dynamic statement>.

Format

```
<using clause> ::=
        <using arguments>
      | <using descriptor>

<using arguments> ::=
      { USING | INTO } <argument> [ { <comma> <argument> }... ]

<argument> ::= <target specification>

<using descriptor> ::=
      { USING | INTO } SQL DESCRIPTOR <descriptor name>
```

Syntax Rules

1) The keyword INTO shall appear in <using clause> only if the <using clause> is contained in a <dynamic fetch statement> or a <result using clause>.

Access Rules

None.

General Rules

1) The <identifier>s used to reference components of SQL descriptor areas are as shown in Table 24, "Data types of <key word>s used in SQL item descriptor areas".

2) If a <descriptor name> is specified in a <using clause>, then an SQL system descriptor area shall have been allocated and not yet deallocated whose name is the value of the <descriptor name>'s <simple value specification> and whose scope is that specified by the <scope option>. Otherwise, an exception condition is raised: *invalid SQL descriptor name*.

3) When a <describe output statement> is executed, a representation of the column descriptors of the <select list> columns for the prepared statement is stored in the specified SQL descriptor area as follows:

   a) Let *N* be the <occurrences> specified when the <descriptor name> was allocated.

   b) If the prepared statement that is being described is a <dynamic select statement> or a <dynamic single row select statement>, then let *T* be the table defined by the prepared statement and let *D* be the degree of *T*. Otherwise, let *D* be 0.

   c) COUNT is set to *D*.

   d) If *D* is greater than *N*, then a completion condition is raised: *warning—insufficient item descriptor areas*.

e) If *D* is 0 or *D* is greater than *N*, then no item descriptor areas are set. Otherwise, the first *D* item descriptor areas are set so that the *i*-th item descriptor area contains the descriptor of the *i*-th column of *T*. The descriptor of a column consists of values for TYPE, NULLABLE, NAME, UNNAMED, and other fields depending on the value of TYPE as described below. The DATA and INDICATOR fields are not relevant in this case. Those fields and fields that are not applicable for a particular value of TYPE are set to implementation-dependent values.

i) TYPE is set to a code, as shown in Table 25, "Codes used for SQL data types in Dynamic SQL", indicating the data type of the column.

ii) NULLABLE is set to 1 if the resulting column is possibly nullable and 0 otherwise.

iii) If the column name is implementation-dependent, then NAME is set to the implementation-dependent name of the column, and UNNAMED is set to 1. Otherwise, NAME is set to the <derived column> name for the column and UNNAMED is set to 0.

iv) If the column does not have the general null class, then the values of NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, and NULL_CLASS_NAME are set to the catalog name, schema name, and null class name of the null class for the column; otherwise they are set to <space>s.

v) Case:

1) If TYPE indicates a <character string type>, then: LENGTH is set to the length or maximum length in characters of the character string; OCTET_LENGTH is set to the maximum possible length in octets of the character string; CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA and CHARACTER_SET_NAME are set to the <character set name> of the character string's character set; and COLLATION_CATALOG, COLLATION_SCHEMA and COLLATION_NAME are set to the <collation name> of the character string's collation. If the subject <language clause> specifies C, then the lengths specified in LENGTH and OCTET_LENGTH do not include the implementation-defined null character that terminates a C character string.

2) If TYPE indicates a <bit string type>, then LENGTH is set to the length or maximum length in bits of the bit string and OCTET_LENGTH is set to the maximum possible length in octets of the bit string.

3) If TYPE indicates an <exact numeric type>, then PRECISION and SCALE are set to the precision and scale of the exact numeric.

4) If TYPE indicates an <approximate numeric type>, then PRECISION is set to the precision of the approximate numeric.

5) If TYPE indicates a <datetime type>, then LENGTH is set to the length in positions of the datetime type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 26, "Codes associated with datetime data types in Dynamic SQL", to indicate the specific datetime data type, and PRECISION is set to the <time precision> or <timestamp precision>, if either is applicable.

6) If TYPE indicates an <interval type>, then DATETIME_INTERVAL_CODE is set to a code as specified in Table 27, "Codes used for <interval qualifier>s in Dynamic SQL", to indicate the <interval qualifier> of the interval data type, DATETIME_INTERVAL_PRECISION is set to the <interval leading field precision>, and

PRECISION is set to the <interval fractional seconds precision>, if applicable.

7) If TYPE indicates the <boolean type>, then PRECISION, SCALE, LENGTH, and OCTET_LENGTH are implementation-dependent.

8) If TYPE indicates an abstract data type, then the values of LENGTH, OCTET_LENGTH, SCALE, and PRECISION are implementation-dependent.

4) When a <describe input statement> is executed, a descriptor for the <dynamic parameter specification>s for the prepared statement is stored in the specified SQL descriptor area as follows:

a) Let *N* be the <occurrences> specified when the <descriptor name> was allocated.

b) Let *D* be the number of <dynamic parameter specification>s in the prepared statement.

c) COUNT is set to *D*.

d) If *D* is greater than *N*, then a completion condition is raised: *warning—insufficient item descriptor areas*.

e) If *D* is 0 or *D* is greater than *N*, then no item descriptor areas are set. Otherwise, the first *D* item descriptor areas are set so that the *i*-th item descriptor area contains a descriptor of the *i*-th <dynamic parameter specification>. The descriptor of a <dynamic parameter specification> consists of values for TYPE, NULLABLE, NAME, UNNAMED, and other fields depending on the value of TYPE as described below. The DATA and INDICATOR fields are not relevant in this case. Those fields and fields that are not applicable for a particular value of TYPE are set to implementation-dependent values.

i) TYPE is set to a code, as shown in Table 25, "Codes used for SQL data types in Dynamic SQL", indicating the data type of the <dynamic parameter specification>.

ii) NULLABLE is set to 1.
Note: This indicates that the <dynamic parameter specification> can have the null value.

iii) UNNAMED is set to 1 and NAME is set to an implementation-dependent value.

iv) The values of NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, and NULL_CLASS_NAME are implementation-dependent.

v) Case:

1) If TYPE indicates a <character string type>, then: LENGTH is set to the length or maximum length in characters of the character string; OCTET_LENGTH is set to the maximum possible length in octets of the character string; CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA and CHARACTER_SET_NAME are set to the <character set name> of the character string's character set; and COLLATION_CATALOG, COLLATION_SCHEMA and COLLATION_NAME are set to the <collation name> of the character string's collation. If the subject <language clause> specifies C, then the lengths specified in LENGTH and OCTET_LENGTH do not include the implementation-defined null character that terminates a C character string.

2) If TYPE indicates a <bit string type>, then LENGTH is set to the length or maximum length in bits of the bit string and OCTET_LENGTH is set to the maximum possible length in octets of the bit string.

3) If TYPE indicates an <exact numeric type>, then PRECISION and SCALE are set to the precision and scale of the exact numeric.

4) If TYPE indicates an <approximate numeric type>, then PRECISION is set to the precision of the approximate numeric.

5) If TYPE indicates a <datetime type>, then LENGTH is set to the length in positions of the datetime type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 26, "Codes associated with datetime data types in Dynamic SQL", to indicate the specific datetime data type and PRECISION is set to the <time precision> or <timestamp precision>, if either is applicable.

6) If TYPE indicates an <interval type>, then DATETIME_INTERVAL_CODE is set to a code as specified in Table 27, "Codes used for <interval qualifier>s in Dynamic SQL", to indicate the <interval qualifier> of the interval data type, DATETIME_INTERVAL_PRECISION is set to the <interval leading field precision> and PRECISION is set to the <interval fractional seconds precision>, if applicable.

7) If TYPE indicates the <boolean type>, then PRECISION, SCALE, LENGTH, and OCTET_LENGTH are set to implementation-dependent values.

8) If TYPE indicates an abstract data type, then the values of LENGTH, OCTET_LENGTH, SCALE, and PRECISION are set to implementation-dependent values.

5) When a <using clause> is used in a <dynamic open statement> or as the <parameter using clause> in an <execute statement>, the <using clause> describes the <dynamic parameter specification> values for the <dynamic open statement> or the <execute statement>, respectively. Let *PS* be the prepared <dynamic select statement> referenced by the <dynamic open statement> or the prepared statement referenced by the <execute statement>, respectively.

   Let *D* be the number of <dynamic parameter specification>s in *PS*.

   a) If <using arguments> is specified and the number of <argument>s is not *D*, then an exception condition is raised: *dynamic SQL error— using clause does not match dynamic parameter specifications*.

   b) If <using descriptor> is specified, then:

      i) If the value of COUNT is greater than the number of <occurrences> specified when the <descriptor name> was allocated or is less than zero, then an exception condition is raised: *dynamic SQL error—invalid descriptor count*.

      ii) If the value of COUNT is not *D*, then an exception condition is raised: *dynamic SQL error— using clause does not match dynamic parameter specifications*.

      iii) If the first *D* item descriptor areas are not valid as specified in Subclause 19.1, "Description of SQL item descriptor areas", then an exception condition is raised: *dynamic SQL error— using clause does not match dynamic parameter specifications*.

iv) If the value of INDICATOR is not negative, and the value of DATA is not a valid value of the data type indicated by TYPE, then an exception condition is raised: *dynamic SQL error— using clause does not match dynamic parameter specifications*.

c) Let *TDT* be the effective data type of the *i*-th <dynamic parameter specification>, defined to be the type represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME that would be set by a <describe input statement> to reflect the descriptor of the *i*-th dynamic parameter of *PS*.

Note: See the General Rules of Subclause 19.8, "<describe statement>".

d) Case:

i) If <using descriptor> is specified, then let *SDT* be the effective data type of the *i*-th <dynamic parameter specification> value as represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME in the *i*-th item descriptor area. If the INDICATOR value of the *i*-th item descriptor area is not negative, then let *SV* be the value represented by the value of DATA with data type *SDT*. Otherwise, let *SV* be the general null value.

ii) If <using arguments> is specified, then let *SDT* and *SV* be the data type and value, respectively, of the *i*-th <argument>.

e) If the <cast specification>

CAST (*SV* AS *TDT*)

violates the Syntax Rules of Subclause 6.13, "<cast specification>", then an exception condition is raised: *dynamic SQL error—restricted data type attribute violation*.

f) If the <cast specification>

CAST (S*V* AS *TDT*)

violates the General Rules of Subclause 6.13, "<cast specification>", then an exception condition is raised in accordance with the General Rules of Subclause 6.13, "<cast specification>".

g) The <cast specification>

CAST (*SV* AS *TDT*)

is effectively performed and is the value of the *i*-th dynamic parameter.

6) When a <using clause> is used in a <dynamic fetch statement> or as the <result using clause> of an <execute statement>, it describes the <target specification>s for the <dynamic fetch statement> or <execute statement>, respectively. Let *PS* be the prepared <dynamic select statement> referenced by the <dynamic fetch statement> or the prepared <dynamic single row select statement> referenced by the <execute statement>, respectively.

Let *D* be the degree of the table specified by *PS*.

a) If <using arguments> is specified and the number of <argument>s is not *D*, then an exception condition is raised: *dynamic SQL error— using clause does not match target specifications*.

b) If <using descriptor> is specified, then:

i) If the value of COUNT is greater than the number of <occurrences> specified when the <descriptor name> was allocated or less than zero, then an exception condition is raised: *dynamic SQL error—invalid descriptor count*.

ii) If COUNT is not equal to *D*, then an exception condition is raised: *dynamic SQL error— using clause does not match target specifications*.

iii) If the first *D* item descriptor areas are not valid as specified in Subclause 19.1, "Description of SQL item descriptor areas", then an exception condition is raised: *dynamic SQL error— using clause does not match target specifications*.

7) When a <using clause> is used in a <dynamic fetch statement> or as the <result using clause> of an <execute statement>, the result is a set of <target specification> values corresponding to the <select list> columns for the retrieved row. If <using descriptor> is specified, then the SQL descriptor area is set. The value of the *i*-th <target specification> is represented in the *i*-th item descriptor area as follows:

a) Let *SDT* be the effective data type of the *i*-th <select list> column, defined to be the type represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_PRECISION, DATETIME_INTERVAL_CODE, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME that would be set by a <describe output statement> to reflect the description of the *i*-th <select list> column. Let *SV* be the value of the <select list> column, with data type *SDT*.

Note: See the General Rules of Subclause 19.8, "<describe statement>".

b) Let *TDT* be the effective data type of the *i*-th <target specification> as represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME, in the *i*-th item descriptor area.

c) If the <cast specification>

CAST (*SV* AS *TDT*)

violates the Syntax Rules of Subclause 6.13, "<cast specification>", then an exception condition is raised: *dynamic SQL error— restricted data type attribute violation*.

d) If the <cast specification>

CAST (*SV* AS *TDT*)

violates the General Rules of Subclause 6.13, "<cast specification>", then an exception condition is raised in accordance with the General Rules of Subclause 6.13, "<cast specification>".

e) The <cast specification>

CAST (*SV* AS *TDT*)

is effectively performed, and is the value *TV* of the *i*-th <target specification>.

f) If *TV* is a null value, then the value of INDICATOR is set to the negative number associated with that null value..

g) If *TV* is not a null value, then:

i) The value of INDICATOR is set to 0 and the value of DATA is set to *TV*.

ii) Case:

1) If TYPE indicates CHARACTER VARYING or BIT VARYING, then RETURNED_LENGTH is set to the length in characters or bits, respectively, of *TV*, and RETURNED_OCTET_LENGTH is set to the length in octets of *TV*.

2) If *SDT* is CHARACTER VARYING or BIT VARYING, then RETURNED_LENGTH is set to the length in characters or bits, respectively, of *SV*, and RETURNED_OCTET_LENGTH is set to the length in octets of *SV*.

Note: All other values of the SQL descriptor area are unchanged.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) A <descriptor name> shall be a <literal>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.10 <execute statement>

Function

Associate input parameters and output targets with a prepared statement and execute the statement.

Format

```
<execute statement> ::=
       EXECUTE <SQL statement name>
         [ <result using clause> ]
         [ <parameter using clause> ]

<result using clause> ::= <using clause>

<parameter using clause> ::= <using clause>
```

Syntax Rules

1) If <SQL statement name> is a <statement name>, then the containing <module> shall contain a <prepare statement> whose <statement name> is the same as the <statement name> of the <execute statement>.

2) If <result using clause> is specified, then the <result using clause> shall contain either a <using arguments> or <using descriptor> that contains the keyword INTO.

Access Rules

None.

General Rules

1) When the <execute statement> is executed, if the <SQL statement name> does not identify a statement *P* previously prepared in the scope of the <SQL statement name>, then an exception condition is raised: *invalid SQL statement name*.

2) If *P* is a <dynamic select statement> that does not conform to the Format and Syntax Rules of a <dynamic single row select statement>, then an exception condition is raised: *dynamic SQL error—cursor specification cannot be executed*.

Editor's Note
Inconsistent use of terms (e.g., <dynamic select statement>/<dynamic single row select statement> or <cursor specification>/<query specification>) has been identified as a Possible Problem. See Possible Problem 155 in the Editor's Notes.

3) If *P* contains the <table name> of a created or declared local temporary table and if the <execute statement> is not in the same <module> as the <prepare statement> that prepared the prepared statement, then an exception condition is raised: *syntax rule or access rule violation in SQL dynamic statement*.

4) If *P* contains <dynamic parameter specification>s and a <parameter using clause> is not specified, then an exception condition is raised: *dynamic SQL error— using clause required for dynamic parameters*.

5) If P is a <dynamic single row select statement> or <call statement> and a <result using clause> is not specified, then an exception condition is raised: *dynamic SQL error—using clause required for result fields*.

6) If a <parameter using clause> that is <using descriptor> is specified, then the General Rules specified in Subclause 19.9, "<using clause>", for a <parameter using clause> in an <execute statement> are applied.

7) If *P* is a <dynamic single row select statement>, then the General Rules specified in Subclause 19.9, "<using clause>", for a <result using clause> in an <execute statement> are applied.

8) *P* is executed.

   Case:

   a) If *P* is a <dynamic single row select statement>, then all General Rules in Subclause 15.5, "<select statement: single row>", apply to *P*, replacing ''<query specification> *S*'' with ''the <query specification> contained in *P*''. The <argument>s contained in the <result using clause> or the item descriptor areas of the SQL descriptor referenced in the <result using clause>, if any, provide the <target specification>s corresponding to the <select list> of *P*.

   b) If the <preparable statement> is a <preparable dynamic delete statement: positioned>, then all General Rules in Subclause 19.19, "<preparable dynamic delete statement: positioned>", apply to the <preparable statement>.

   c) If the <preparable statement> is a <preparable dynamic update statement: positioned>, then all General Rules in Subclause 19.20, "<preparable dynamic update statement: positioned>", apply to the <preparable statement>.

   d) Otherwise, results of the execution are the same as if the statement statement was contained in a <routine> and executed; these are described in Subclause 13.2, "<routine>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <result using clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.11 <execute immediate statement>

Function

Dynamically prepare and execute a preparable statement.

Format

```
<execute immediate statement> ::=
       EXECUTE IMMEDIATE <SQL statement variable>
```

Syntax Rules

1) The data type of <SQL statement variable> shall be character string.

Access Rules

None.

General Rules

1) Let *P* be the contents of the <SQL statement variable>.

2) If *P* is a <preparable dynamic delete statement: positioned> or a <preparable dynamic update statement: positioned>, then *P* refers to either a dynamic cursor with the same <cursor name> or to an extended dynamic cursor whose <extended cursor name> value is the same as the <cursor name>.

   Case:

   a) If both an extended dynamic cursor and a dynamic cursor with the same name as the <cursor name> exist, then an exception condition is raised: *ambiguous cursor name*.

   b) If there is neither an extended dynamic cursor nor a dynamic cursor with the name of <cursor name>, then an exception condition is raised: *invalid cursor name*.

3) If one or more of the following are true, then an exception condition is raised: *syntax error or access rule violation in dynamic SQL statement*.

   a) *P* does not conform to the Format, Syntax Rules, and Access Rules for a <preparable statement> or *P* is a <dynamic select statement> or a <dynamic single row select statement>.

   b) *P* contains a <simple comment>.

   c) *P* contains a <dynamic parameter specification>.

4) The <preparable statement> that is the value of the <SQL statement variable> is prepared and executed.

Case:

a) If the <preparable statement> is a <preparable dynamic delete statement: positioned>, then all General Rules in Subclause 19.19, "<preparable dynamic delete statement: positioned>", apply to the <preparable statement>.

b) If the <preparable statement> is a <preparable dynamic update statement: positioned>, then all General Rules in Subclause 19.20, "<preparable dynamic update statement: positioned>", apply to the <preparable statement>.

c) Otherwise, the results of the execution are the same as if the statement was contained in a <routine> and executed; these are described in Subclause 13.2, "<routine>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.12 <dynamic declare cursor>

Function

Declare a cursor to be associated with a <statement name>, which may in turn be associated with a <cursor specification>.

Format

```
<dynamic declare cursor> ::=
       DECLARE <cursor name> [ INSENSITIVE ] [ SCROLL ] CURSOR FOR <statement name>
```

Syntax Rules

1) The <cursor name> shall not be identical to the <cursor name> specified in any other <declare cursor> or <dynamic declare cursor> in the same <module>.

2) The containing <module> shall contain a <prepare statement> whose <statement name> is the same as the <statement name> of the <dynamic declare cursor>.

Access Rules

None.

General Rules

1) All General Rules of Subclause 15.1, "<declare cursor>" apply to <dynamic declare cursor>, replacing ''<open statement>'' with ''<dynamic open statement>'' and ''<cursor specification>'' with ''prepared statement''.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <dynamic declare cursor> that specifies INSENSITIVE.

   b) If an <updatability clause> of FOR UPDATE with or without a <column name list> is specified, then neither SCROLL nor ORDER BY shall be specified.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.13 <allocate cursor statement>

Function

Define a cursor based on a <prepare statement> for a <cursor specification>.

Format

```
<allocate cursor statement> ::=
       ALLOCATE <extended cursor name> [INSENSITIVE ] [ SCROLL ] CURSOR
         FOR <extended statement name>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) When the <allocate cursor statement> is executed, if the value of the <extended statement name> does not identify a statement previously prepared in the scope of the <extended statement name>, then an exception condition is raised: *invalid SQL statement name*.

2) If the prepared statement associated with the <extended statement name> is not a <cursor specification>, then an exception condition is raised: *dynamic SQL error—prepared statement not a cursor specification*.

3) All General Rules of Subclause 15.1, "<declare cursor>" apply to <allocate cursor statement>, replacing ''<open statement>'' with ''<dynamic open statement>'' and ''<cursor specification>'' with ''prepared statement''.

4) Let *S* be the character string that is the value of the <simple value specification> immediately contained in <extended cursor name>. Let *V* be the character string that is the result of

   TRIM (BOTH ' ' FROM *S*)

   If *V* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid cursor name*.

5) If the value of the <extended cursor name> is identical to the value of the <extended cursor name> of any other cursor allocated in the scope of the <extended cursor name>, then an exception condition is raised: *invalid cursor name*.

6) An association is made between the value of the <extended cursor name> and the prepared statement in the scope of the <extended cursor name>. The association is preserved until the prepared statement is destroyed, at which time the cursor identified by <extended cursor name> is also destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <allocate cursor statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

19.14 <dynamic open statement>

Function

Associate input parameters with a <cursor specification> and open the cursor.

Format

```
<dynamic open statement> ::=
       OPEN <dynamic cursor name> [ <using clause> ]
         [ <open cascade option> ]
```

Syntax Rules

1) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <module> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

2) Let *CR* be the cursor identified by *DCN*.

Access Rules

1) The Access Rules for the <query expression> simply contained in the prepared statement associated with the <dynamic cursor name> are applied.

General Rules

1) If <dynamic cursor name> is a <cursor name> and the <statement name> of the associated <dynamic declare cursor> is not associated with a prepared statement, then an exception condition is raised: *invalid SQL statement name*.

2) If <dynamic cursor name> is an <extended cursor name> whose value does not identify a cursor allocated in the scope of the <extended cursor name>, then an exception condition is raised: *invalid cursor name*.

3) If the prepared statement associated with the <dynamic cursor name> contains <dynamic parameter specification>s and a <using clause> is not specified, then an exception condition is raised: *dynamic SQL error—using clause required for dynamic parameters*.

4) The cursor specified by <dynamic cursor name> is updatable if and only if the associated <cursor specification> specified an updatable cursor.

   Note: ‘‘updatable cursor’’ is defined in Subclause 15.1, "<declare cursor>".

5) If a <using clause> is specified, then the General Rules specified in Subclause 19.9, "<using clause>", for <dynamic open statement> are applied.

6) All General Rules of Subclause 15.2, "<open statement>", apply to the <dynamic open statement>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.15 <dynamic fetch statement>

Function

Fetch a row for a cursor declared with a <dynamic declare cursor>.

Format

```
<dynamic fetch statement> ::=
       FETCH [ [ <fetch orientation> ] FROM ] <dynamic cursor name> <using clause>
```

Syntax Rules

1) If <fetch orientation> is omitted, then NEXT is implicit.

2) The <using clause> shall specify INTO.

3) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <module> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

4) Let *CR* be the cursor identified by *DCN* and let *T* be the table defined by the <cursor specification> of *CR*.

5) If the implicit or explicit <fetch orientation> is not NEXT, then the <dynamic declare cursor> or <allocate cursor statement> associated with *CR* shall specify SCROLL.

6) The number of <target specification>s in <using arguments> or the number of item descriptor areas in the SQL descriptor area referenced by <using descriptor>, as appropriate, shall be the same as the degree of *T*. The *i*-th <target specification> in <using arguments> or the *i*-th item descriptor area of the SQL descriptor area, as appropriate, corresponds with the *i*-th column of *T*.

7) The Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to each corresponding <target specification> in <using arguments> and each column of *T* as *TARGET* and *VALUE*, respectively.

Access Rules

None.

General Rules

1) The General Rules specified in Subclause 19.9, "<using clause>", for <dynamic fetch statement> are applied.

2) All General Rules of Subclause 15.3, "<fetch statement>", apply to the <dynamic fetch statement>, replacing ‘‘targets in the <fetch target list>’’ and ‘‘targets identified by the <fetch target list>’’ with ‘‘<target specification>s in the <using arguments> or item descriptor areas of the SQL descriptor area, as appropriate’’.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.16 <dynamic close statement>

Function

Close a cursor.

Format

```
<dynamic close statement> ::=
       CLOSE <dynamic cursor name>
```

Syntax Rules

1) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <module> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

2) Let *CR* be the cursor identified by *DCN*.

Access Rules

None.

General Rules

1) All General Rules of Subclause 15.4, "<close statement>", apply to the <dynamic close statement>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.17 <dynamic delete statement: positioned>

Function

Delete a row of a table.

Format

```
<dynamic delete statement: positioned> ::=
       DELETE FROM <table name>
         WHERE CURRENT OF <dynamic cursor name>
```

Syntax Rules

1) If <dynamic cusror name> *DCN* is a <cursor name> *CN*, then the containing <module> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

2) Let *CR* be the cursor identified by *DCN*.

3) *CR* shall be an updatable cursor.

   Note: ''updatable cursor'' is defined in Subclause 15.1, "<declare cursor>".

4) Let *T* be the table identified by the <table name>. Let *QS* be the <query specification> that is the simply underlying table of the simply underlying table of *CR*. The simply underlying table of *QS* shall be *T*.

   Note: The ''simply underlying table'' of a <cursor specification> is defined in Subclause 15.1, "<declare cursor>".

Access Rules

1) All Access Rules of Subclause 15.6, "<delete statement: positioned>", apply to the <dynamic delete statement: positioned>.

General Rules

1) All General Rules of Subclause 15.6, "<delete statement: positioned>", apply to the <dynamic delete statement: positioned>, replacing ''<delete statement: positioned>'' with ''<dynamic delete statement: positioned>''.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.18 <dynamic update statement: positioned>

Function

Update a row of a table.

Format

```
<dynamic update statement: positioned> ::=
       UPDATE <table name>
         SET <set clause list>
           WHERE CURRENT OF <dynamic cursor name>
```

Syntax Rules

1) If <dynamic cusror name> *DCN* is a <cursor name> *CN*, then the containing <module> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

2) Let *CR* be the cursor identified by *DCN*.

3) *CR* shall be an updatable cursor.

   Note: ''updatable cursor'' is defined in Subclause 15.1, "<declare cursor>".

4) Let *T* be the table identified by the <table name>. Let *QS* be the <query specification> that is the simply underlying table of the simply underlying table of *CR*. The simply underlying table of *QS* shall be *T*.

   Note: The ''simply underlying table'' of a <cursor specification> is defined in Subclause 15.1, "<declare cursor>".

5) If *CR* is an ordered cursor, then for each <object column> *OC*, the column of *T* identified by *OC* shall not be directly or indirectly referenced in the <order by clause> of the defining <cursor specification> for *CR*.

6) No leaf generally underlying table of *T* shall be an underlying table of any <query expression> generally contained in any <value expression> immediately contained in any <update source> contained in the <set clause list>.

• 2 Rules deleted.

7) If *CR* was specified using an explicit or implicit <updatability clause> of FOR UPDATE, then each <column name> specified as an <object column> shall identify a column in the explicit or implicit <column name list> associated with the <updatability clause>.

8) The scope of the <table name> is the entire <update statement: positioned>.

• 1 Rule deleted.

Access Rules

1) All Access Rules of Subclause 15.9, "<update statement: positioned>", apply to the <dynamic update statement: positioned>.

General Rules

1) All General Rules of Subclause 15.9, "<update statement: positioned>", apply to the <dynamic update statement: positioned>, replacing ‘‘<cursor name>’’ with ‘‘<dynamic cursor name>’’ and ‘‘<update statement: positioned>’’ with ‘‘<dynamic update statement: positioned>’’.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

19.19 <preparable dynamic delete statement: positioned>

Function

Delete a row of a table through a dynamic cursor.

Format

```
<preparable dynamic delete statement: positioned> ::=
       DELETE [ FROM <table name> ]
         WHERE CURRENT OF <cursor name>
```

Syntax Rules

1) If <table name> is not specified, then the name of the underlying table of the <cursor specification> identified by <cursor name> is implicit.

2) All Syntax Rules of Subclause 15.6, "<delete statement: positioned>", apply to the <preparable dynamic delete statement: positioned>, replacing ''<declare cursor>'' with ''<dynamic declare cursor> or <allocate cursor statement>'' and ''<delete statement: positioned>'' with ''<preparable dynamic delete statement: positioned>''.

Access Rules

1) All Access Rules of Subclause 15.6, "<delete statement: positioned>", apply to the <preparable dynamic delete statement: positioned>.

General Rules

1) All General Rules of Subclause 15.6, "<delete statement: positioned>", apply to the <preparable dynamic delete statement: positioned>, replacing ''<delete statement: positioned>'' with ''<preparable dynamic delete statement: positioned>''.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <preparable dynamic delete statement: positioned>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None. language.

19.20 <preparable dynamic update statement: positioned>

Function

Update a row of a table through a dynamic cursor.

Format

```
<preparable dynamic update statement: positioned> ::=
       UPDATE [ <table name> ]
         SET <set clause list>
         WHERE CURRENT OF <cursor name>
```

Syntax Rules

1) If <table name> is not specified, then the name of the underlying table of the <cursor specification> identified by <cursor name> is implicit.

2) All Syntax Rules of Subclause 15.9, "<update statement: positioned>", apply to the <preparable dynamic update statement: positioned>, replacing ''<declare cursor>'' with ''<dynamic declare cursor> or <allocate cursor statement>'' and ''<update statement: positioned>'' with ''<preparable dynamic update statement: positioned>''.

Access Rules

1) All Access Rules of Subclause 15.9, "<update statement: positioned>", apply to the <preparable dynamic update statement: positioned>.

General Rules

1) All General Rules of Subclause 15.9, "<update statement: positioned>", apply to the <preparable dynamic update statement: positioned>, replacing ''<update statement: positioned>'' with ''<preparable dynamic update statement: positioned>''.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <preparable dynamic update statement: positioned>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

20 Diagnostics management

20.1 <get diagnostics statement>

Function

Get exception or completion condition information from the diagnostics area.

Format

```
<get diagnostics statement> ::=
       GET DIAGNOSTICS <sql diagnostics information>

<sql diagnostics information> ::=
         <statement information>
       | <condition information>

<statement information> ::=
       <statement information item> [ { <comma> <statement information item> }... ]

<statement information item> ::=
       <simple target specification> <equals operator> <statement information item name>


<statement information item name> ::=
         NUMBER
       | MORE
       | COMMAND_FUNCTION
       | DYNAMIC_FUNCTION
       | ROW_COUNT

<condition information> ::=
       EXCEPTION <condition number>
         <condition information item> [ { <comma> <condition information item> }... ]

<condition information item> ::=
       <simple target specification> <equals operator> <condition information item name>


<condition information item name> ::=
         CONDITION_NUMBER
       | RETURNED_SQLSTATE
       | CLASS_ORIGIN
       | SUBCLASS_ORIGIN
       | SERVER_NAME
       | CONNECTION_NAME
       | CONSTRAINT_CATALOG
       | CONSTRAINT_SCHEMA
       | CONSTRAINT_NAME
       | TRIGGER_CATALOG
       | TRIGGER_SCHEMA
       | TRIGGER_NAME
       | CATALOG_NAME
       | SCHEMA_NAME
       | TABLE_NAME
```

```
        | COLUMN_NAME
        | CURSOR_NAME
```

ANSI Only--SQL3

```
        | ROUTINE_CATALOG
        | ROUTINE_SCHEMA
        | ROUTINE_NAME
        | SPECIFIC_NAME
```

```
        | MESSAGE_TEXT
        | MESSAGE_LENGTH
        | MESSAGE_OCTET_LENGTH
```

```
<condition number> ::= <simple value specification>
```

Syntax Rules

1) The data type of a <simple target specification> contained in a <statement information item> or <condition information item> shall be the data type specified in Table 28, "<identifier>s for use with <get diagnostics statement>", for the corresponding <statement information item name> or <condition information item name>.

2) The data type of <condition number> shall be exact numeric with scale 0.

Table 28—<identifier>s for use with <get diagnostics statement>

<identifier>	Data Type
<statement information item name>s	
NUMBER	exact numeric with scale 0
MORE	character string (1)
COMMAND_FUNCTION	character varying (*L*)
DYNAMIC_FUNCTION	character varying (*L*)
ROW_COUNT	exact numeric with scale 0
<condition information item name>s	
CONDITION_NUMBER	exact numeric with scale 0
RETURNED_SQLSTATE	character string (5)
CLASS_ORIGIN	character varying (*L*)
SUBCLASS_ORIGIN	character varying (*L*)
SERVER_NAME	character varying (*L*)
CONNECTION_NAME	character varying (*L*)
CONSTRAINT_CATALOG	character varying (*L*)
CONSTRAINT_SCHEMA	character varying (*L*)
CONSTRAINT_NAME	character varying (*L*)
TRIGGER_CATALOG	character varying (*L*)
TRIGGER_SCHEMA	character varying (*L*)
TRIGGER_NAME	character varying (*L*)
CATALOG_NAME	character varying (*L*)
SCHEMA_NAME	character varying (*L*)
TABLE_NAME	character varying (*L*)
COLUMN_NAME	character varying (*L*)
CURSOR_NAME	character varying (*L*)

ANSI Only–SQL3

ROUTINE_CATALOG	character varying (*L*)
ROUTINE_SCHEMA	character varying (*L*)
ROUTINE_NAME	character varying (*L*)
SPECIFIC_NAME	character varying (*L*)

Where *L* is an implementation-defined integer not less than 128.

(Continued on next page)

Table 28—<identifier>s for use with <get diagnostics statement> (Cont.)

<identifier>	Data Type
<condition information item name>s	
MESSAGE_TEXT	character varying (*L*)
MESSAGE_LENGTH	exact numeric with scale 0
MESSAGE_OCTET_LENGTH	exact numeric with scale 0

Access Rules

None.

General Rules

1) Specification of <statement information item> retrieves information about the statement execution recorded in the diagnostics area into <simple target specification>.

   a) The value of NUMBER is the number of exception or completion conditions that have been stored in the diagnostics area as a result of executing the previous SQL-statement other than a <get diagnostics statement>.

      Note: The <get diagnostics statement> itself may return information via the SQLCODE or SQLSTATE parameters, but does not modify the previous contents of the diagnostics area.

   b) The value of MORE is:

      Y — More conditions were raised during execution of the SQL-statement than have been stored in the diagnostics area.

      N — All of the conditions that were raised during execution of the SQL-statement have been stored in the diagnostics area.

   c) The value of COMMAND_FUNCTION is the identification of the SQL-statement executed. Table 29, "SQL-statement character codes for use in the diagnostics area" specifies the identifier of the SQL-statements.

   d) The value of DYNAMIC_FUNCTION is the identification of the prepared statement executed. Table 29, "SQL-statement character codes for use in the diagnostics area", specifies the identifier of the SQL-statements.

Table 29—SQL-statement character codes for use in the diagnostics area

SQL-statement	Identifier
<allocate cursor statement>	ALLOCATE CURSOR
<allocate descriptor statement>	ALLOCATE DESCRIPTOR
<alter domain statement>	ALTER DOMAIN
<alter table statement>	ALTER TABLE
<assignment statement>	ASSIGNMENT
<assertion definition>	CREATE ASSERTION

Table 29—SQL-statement character codes for use in the diagnostics area (Cont.)

SQL-statement	Identifier
<call statement>	CALL
<character set definition>	CREATE CHARACTER SET
<close statement>	CLOSE CURSOR
<collation definition>	CREATE COLLATION
<commit statement>	COMMIT WORK
<compound statement>	BEGIN END
<connect statement>	CONNECT
<deallocate descriptor statement>	DEALLOCATE DESCRIPTOR
<deallocate prepared statement>	DEALLOCATE PREPARE
<destroy statement>	DESTROY
<external function declaration>	DECLARE EXTERNAL FUNCTION
<delete statement: positioned>	DELETE CURSOR
<delete statement: searched>	DELETE WHERE
<describe statement>	DESCRIBE
<direct select statement: multiple rows>	SELECT
<disconnect statement>	DISCONNECT
<domain definition>	CREATE DOMAIN
<drop assertion statement>	DROP ASSERTION
<drop character set statement>	DROP CHARACTER SET
<drop collation statement>	DROP COLLATION
<drop domain statement>	DROP DOMAIN
<drop external function statement>	DROP EXTERNAL FUNCTION
<drop module statement>	DROP MODULE
<drop role statement>	DROP ROLE
<drop schema statement>	DROP SCHEMA
<drop table statement>	DROP TABLE
<drop translation statement>	DROP TRANSLATION
<drop trigger statement>	DROP TRIGGER
<drop data type statement>	DROP DATA TYPE
<drop view statement>	DROP VIEW
<dynamic close statement>	DYNAMIC CLOSE
<dynamic delete statement: positioned>	DYNAMIC DELETE CURSOR
<dynamic fetch statement>	DYNAMIC FETCH
<dynamic open statement>	DYNAMIC OPEN

Table 29—SQL-statement character codes for use in the diagnostics area (Cont.)

SQL-statement	Identifier
<dynamic single row select statement>	SELECT
<dynamic update statement: positioned>	DYNAMIC UPDATE CURSOR
<execute immediate statement>	EXECUTE IMMEDIATE
<execute statement>	EXECUTE
<fetch statement>	FETCH
<get descriptor statement>	GET DESCRIPTOR
<grant statement>	GRANT
<grant role statement>	GRANT ROLE
<insert statement>	INSERT
<module definition>	DEFINE MODULE
<new statement>	NEW
<open statement>	OPEN
<preparable dynamic delete statement: positioned>	DYNAMIC DELETE CURSOR
<preparable dynamic update statement: positioned>	DYNAMIC UPDATE CURSOR
<prepare statement>	PREPARE
<release savepoint statement>	RELEASE SAVEPOINT
<return statement>	RETURN
<revoke statement>	REVOKE
<revoke role statement>	REVOKE ROLE
<role definition>	CREATE ROLE
<rollback statement>	ROLLBACK WORK
<savepoint statement>	SAVEPOINT
<schema definition>	CREATE SCHEMA
<select statement: single row>	SELECT
<set catalog statement>	SET CATALOG
<set connection statement>	SET CONNECTION
<set constraints mode statement>	SET CONSTRAINT
<set descriptor statement>	SET DESCRIPTOR
<set local time zone statement>	SET TIME ZONE
<set names statement>	SET NAMES
<set role statement>	SET ROLE
<set schema statement>	SET SCHEMA

Table 29—SQL-statement character codes for use in the diagnostics area (Cont.)

SQL-statement	Identifier
<set transaction statement>	SET TRANSACTION
<set session authorization identifier statement>	SET SESSION AUTHORIZATION
<table definition>	CREATE TABLE
<test completion statement>	TEST COMPLETION
<translation definition>	CREATE TRANSLATION
<trigger definition>	CREATE TRIGGER
<update statement: positioned>	UPDATE CURSOR
<update statement: searched>	UPDATE WHERE
<abstract data type definition>	CREATE TYPE
<view definition>	CREATE VIEW

e) The value of ROW_COUNT is the number of rows affected as the result of executing a <delete statement: searched>, <insert statement>, or <update statement: searched> as a direct result of executing the previous SQL-statement. Let *S* be the <delete statement: searched>, <insert statement>, or <update statement: searched>. Let *T* be the table identified by the <table name> directly contained in *S*.

Case:

i) If *S* is an <insert statement>, then the value of ROW_COUNT is the number of rows inserted into *T*.

ii) If *S* is not an <insert statement> and does not contain a <search condition>, then the value of ROW_COUNT is the cardinality of *T* before the execution of *S*.

iii) Otherwise, let *SC* be the <search condition> directly contained in *S*. The value of ROW_COUNT is effectively derived by executing the statement:

SELECT COUNT(*) FROM *T* WHERE *SC*

before the execution of *S*.

The value of ROW_COUNT following the execution of an SQL-statement that does not directly result in the execution of a <delete statement: searched>, an <insert statement>, or an <update statement: searched> is implementation-dependent.

2) If <condition information> was specified, then let *N* be the value of <condition number>. If *N* is less than 1 or greater than the number of conditions stored in the diagnostics area, then an exception condition is raised: *invalid condition number*. If <condition number> has the value 1, then the diagnostics information retrieved corresponds to the condition indicated by the SQLSTATE or SQLCODE value actually returned by execution of the previous SQL-statement other than a <get diagnostics statement>. Otherwise, the association between <condition number> values and specific conditions raised during evaluation of the General Rules for that SQL-statement is implementation-dependent.

3) Specification of <condition information item> retrieves information about the *N*-th condition in the diagnostics area into the <simple target specification>.

a) The value of CONDITION_NUMBER is the value of <condition number>.

b) The value of CLASS_ORIGIN is the identification of the naming authority that defined the class value of RETURNED_SQLSTATE. That value shall be 'ISO 9075' for any RETURNED_SQLSTATE whose class value is fully defined in Subclause 24.1, "SQLSTATE", and shall be an implementation-defined character string other than 'ISO 9075' for any RETURNED_SQLSTATE whose class value is an implementation-defined class value.

c) The value of SUBCLASS_ORIGIN is the identification of the naming authority that defined the subclass value of RETURNED_SQLSTATE. That value shall be 'ISO 9075' for any RETURNED_SQLSTATE whose subclass value is fully defined in Subclause 24.1, "SQLSTATE", and shall be an implementation-defined character string other than 'ISO 9075' for any RETURNED_SQLSTATE whose subclass value is an implementation-defined subclass value.

d) The value of RETURNED_SQLSTATE is the SQLSTATE parameter that would have been returned if this were the only completion or exception condition possible.

e) If the value of RETURNED_SQLSTATE corresponds to *warning* with a subclass of *cursor operation conflict*, then the value of CURSOR_NAME is the name of the cursor that caused the completion condition to be raised.

f) If the value of RETURNED_SQLSTATE corresponds to *integrity constraint violation*, *transaction rollback—integrity constraint violation*, or a *triggered data change violation* that was caused by a violation of a referential constraint, then:

   i) The values of CONSTRAINT_CATALOG and CONSTRAINT_SCHEMA are the <catalog name> and the <unqualified schema name> of the <schema name> of the schema containing the constraint or assertion. The value of CONSTRAINT_NAME is the <qualified identifier> of the constraint or assertion.

   ii) Case:

      1) If the violated integrity constraint is a table constraint, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name>, and the <qualified identifier> or <local table name>, respectively, of the table in which the table constraint is contained.

      2) If the violated integrity constraint is an assertion and if only one table referenced by the assertion has been modified as a result of executing the SQL-statement, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name>, and the <qualified identifier> or <local table name>, respectively, of the modified table.

      3) Otherwise, the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are <space>s.

   If TABLE_NAME identifies a declared local temporary table, then CATALOG_NAME is <space>s and SCHEMA_NAME is ''MODULE''.

g) If the value of RETURNED_SQLSTATE corresponds to *triggered action exception*, *transaction rollback—triggered action exception*, or a *triggered data change violation* that was caused by a trigger, then:

   i) The values of TRIGGER_CATALOG and TRIGGER_SCHEMA are the <catalog name> and the <unqualified schema name> of the <schema name> of the schema containing

the trigger. The value of TRIGGER_NAME is the <qualified identifier> of the <trigger name> of the trigger.

ii) The values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name>, and the <qualified identifier> of the <table name>, respectively, of the table on which the trigger is defined.

h) If the value of RETURNED_SQLSTATE corresponds to *syntax error or access rule violation*, *syntax error or access rule violation in dynamic SQL statement*, or *syntax error or access rule violation in direct SQL statement*, then:

i) The values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name> of the schema that contains the table that caused the syntax error or access rule violation and the
ANSI <qualified identifier>,
ISO <qualified identifier> or <local table name>,
respectively. If TABLE_NAME refers to a declared local temporary table, then CATALOG_NAME is <space>s and SCHEMA_NAME contains ''MODULE''.

ii) If the syntax error or access rule violation was for an inaccessible column, then the value of COLUMN_NAME is the <column name> of that column. Otherwise, the value of COLUMN_NAME is <space>s.

i) If the value of RETURNED_SQLSTATE corresponds to *invalid cursor state*, then the value of CURSOR_NAME is the name of the cursor that is in the invalid state.

j) If the value of RETURNED_SQLSTATE corresponds to *with check option violation*, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name> of the schema that contains the view that caused the violation of the WITH CHECK OPTION, and the <qualified identifier> of that view, respectively.

k) If the value of RETURNED_SQLSTATE does not correspond to *syntax error or access rule violation*, *syntax error or access rule violation in dynamic SQL statement*, or *syntax error or access rule violation in direct SQL statement*, then:

i) If the values of CATALOG_NAME, SCHEMA_NAME, TABLE_NAME, and COLUMN_NAME identify a column for which no privileges are granted to the current <authorization identifier> or to the enabled applicable roles of that <authorization identifier>, then the value of COLUMN_NAME is replaced by a zero-length string.

ii) If the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME identify a table for which no privileges are granted to the current <authorization identifier> or to the enabled applicable roles of that <authorization identifier>, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are replaced by a zero-length string.

iii) If the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME identify a <table constraint> for some table *T* and if no privileges for *T* are granted to the current <authorization identifier> or to the enabled applicable roles of that <authorization identifier>, then the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are replaced by a zero-length string.

iv) If the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME identify an assertion contained in some schema *S* and if the owner of *S* is not the <authorization identifier> of the current SQL-session, then the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are replaced by a zero-length string.

ANSI Only–SQL3

l) If the value of RETURNED_SQLSTATE corresponds to *external function call exception* or *external call exception*, then;

i) The values of ROUTINE_CATALOG and ROUTINE_SCHEMA are the <catalog name> and the <unqualified schema name>, respectively, of the <schema name> of the schema containing the routine.

ii) The values of ROUTINE_NAME and SPECIFIC_NAME are the <identifier> of the <routine name> and the <identifier> of the <specific name> of the routine, respectively.

m) The value of MESSAGE_TEXT is an implementation-defined character string.

Note: An implementation may set this to <space>s, to a zero-length string, or to a character string describing the condition indicated by RETURNED_SQLSTATE.

n) The value of MESSAGE_LENGTH is the length in characters of the character string value in MESSAGE_TEXT.

o) The value of MESSAGE_OCTET_LENGTH is the length in octets of the character string value in MESSAGE_TEXT.

p) The values of CONNECTION_NAME and SERVER_NAME are

Case:

i) If COMMAND_FUNCTION or DYNAMIC_FUNCTION identifies an <SQL connection statement>, then the explicit or implicit <connection name> and the associated <SQL-server name>, respectively, referenced by the <SQL connection statement>.

ii) Otherwise, the explicit or implicit <connection name> and the associated <SQL-server name>, respectively, corresponding to the most-recently executed explicit or implicit <connect statement> or <set connection statement>.

q) The values of character string items where not otherwise specified by the preceding rules are set to a zero-length string.

Note: There are no numeric items that are not set by these rules.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any <get diagnostics statement>.

21 Embedded SQL

21.1 <embedded SQL host program>

Function

Specify an <embedded SQL host program>.

Format

```
<embedded SQL host program> ::=
          <embedded SQL Ada program>
        | <embedded SQL C program>
        | <embedded SQL COBOL program>
        | <embedded SQL Fortran program>
        | <embedded SQL MUMPS program>
        | <embedded SQL Pascal program>
        | <embedded SQL PL/I program>

<embedded SQL statement> ::=
        <SQL prefix>
          <statement or declaration>
        [ <SQL terminator> ]

<statement or declaration> ::=
          <declare cursor>
        | <dynamic declare cursor>
        | <temporary table declaration>
        | <temporary view definition>
        | <embedded exception declaration>
```

ANSI Only--SQL3

```
        | <SQL variable declaration>
```

```
        | <routine>
        | <SQL procedure statement>

<SQL prefix> ::=
          EXEC SQL
        | <ampersand>SQL<left paren>

<SQL terminator> ::=
          END-EXEC
        | <semicolon>
        | <right paren>

<embedded SQL declare section> ::=
          <embedded SQL begin declare>
            [ <embedded character set declaration> ]
            [ <host variable definition>... ]
          <embedded SQL end declare>
        | <embedded SQL MUMPS declare>
```

```
<embedded character set declaration> ::=
       SQL NAMES ARE <character set specification>

<embedded SQL begin declare> ::=
       <SQL prefix> BEGIN DECLARE SECTION [ <SQL terminator> ]

<embedded SQL end declare> ::=
       <SQL prefix> END DECLARE SECTION [ <SQL terminator> ]

<embedded SQL MUMPS declare> ::=
       <SQL prefix>
         BEGIN DECLARE SECTION
           [ <embedded character set declaration> ]
           [ <host variable definition>... ]
         END DECLARE SECTION
       <SQL terminator>

<host variable definition> ::=
         <Ada variable definition>
       | <C variable definition>
       | <COBOL variable definition>
       | <Fortran variable definition>
       | <MUMPS variable definition>
       | <Pascal variable definition>
       | <PL/I variable definition>

<embedded variable name> ::=
       <colon><host identifier>

<host identifier> ::=
         <Ada host identifier>
       | <C host identifier>
       | <COBOL host identifier>
       | <Fortran host identifier>
       | <MUMPS host identifier>
       | <Pascal host identifier>
       | <PL/I host identifier>
```

Syntax Rules

1) An <embedded SQL host program> is a compilation unit that consists of programming language text and SQL text. The programming language text shall conform to the requirements of a specific standard programming language. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s, as defined in this
   ANSI American
   ISO International
   Standard.

   Note: ''Compilation unit'' is defined in Subclause 4.25, "Modules".

2) Case:

   a) An <embedded SQL statement> or <embedded SQL MUMPS declare> that is contained in an <embedded SQL MUMPS program> shall contain an <SQL prefix> that is ''<ampersand>SQL<left paren>''. There shall be no <separator> between the <ampersand> and ''SQL'' nor between ''SQL'' and the <left paren>.

b) An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> that is not contained in an <embedded SQL MUMPS program> shall contain an <SQL prefix> that is ''EXEC SQL''.

3) Case:

a) An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> contained in an <embedded SQL COBOL program> shall contain an <SQL terminator> that is END-EXEC.

b) An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> contained in an <embedded SQL Fortran program> shall not contain an <SQL terminator>.

c) An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> contained in an <embedded SQL Ada program>, <embedded SQL C program>, <embedded SQL Pascal program>, or <embedded SQL PL/I program> shall contain an <SQL terminator> that is a <semicolon>.

d) An <embedded SQL statement> or <embedded SQL MUMPS declare> that is contained in an <embedded SQL MUMPS program> shall contain an <SQL terminator> that is a <right paren>.

4) Case:

a) An <embedded SQL declare section> that is contained in an <embedded SQL MUMPS program> shall be an <embedded SQL MUMPS declare>.

b) An <embedded SQL declare section> that is not contained in an <embedded SQL MUMPS program> shall not be an <embedded SQL MUMPS declare>.

Note: There is no restriction on the number of <embedded SQL declare section>s that may be contained in an <embedded SQL host program>.

5) The <token>s comprising an <SQL prefix>, <embedded SQL begin declare>, or <embedded SQL end declare> shall be separated by <space> characters and shall be specified on one line. Otherwise, the rules for the continuation of lines and tokens from one line to the next and for the placement of host language comments are those of the programming language of the containing <embedded SQL host program>.

6) If an <embedded character set declaration> is not specified, then an <embedded character set declaration> that specifies an implementation-defined character set that contains at least every character that is in <SQL language character> is implicit.

• 1 Rule deleted

7) A <temporary table declaration> or <temporary view definition> that is contained in an <embedded SQL host program> shall precede in the text of that <embedded SQL host program> any SQL-statement or <declare cursor> that references the <table name> of the <temporary view definition> or <temporary table declaration>.

8) A <declare cursor> that is contained in an <embedded SQL host program> shall precede in the text of that <embedded SQL host program> any SQL-statement that references the <cursor name> of the <declare cursor>.

9) A <dynamic declare cursor> that is contained in an <embedded SQL host program> shall precede in the text of that <embedded SQL host program> any SQL-statement that references the <cursor name> of the <dynamic declare cursor>.

ANSI Only–SQL3

10) An <SQL variable declaration> that is contained in an <embedded SQL host program> shall precede in the text of that <embedded SQL host program> any SQL-statement that references the <SQL variable name> of the <SQL variable declaration>.

11) An <SQL variable name> contained in an <SQL variable declaration> that is immediately contained in an <embedded SQL host program> shall be different from any other <SQL variable name> or <embedded variable name> contained in any other <SQL variable declaration> or <host variable definition>, respectively, that is immediately contained in the <embedded SQL host program>.

12) Any <host identifier> that is contained in an <embedded SQL statement> in an <embedded SQL host program> shall be defined in exactly one <host variable definition> contained in that <embedded SQL host program>. In programming languages that support <host variable definition>s in subprograms, two <host variable definition>s with different, non-overlapping scope in the host language are to be regarded as defining different host variables, even if they specify the same variable name. That <host variable definition> shall appear in the text of the <embedded SQL host program> prior to any <embedded SQL statement> that references the <host identifier>. The <host variable definition> shall be such that a host language reference to the <host identifier> is valid at every <embedded SQL statement> that contains the <host identifier>.

13) A <host variable definition> defines the host language data type of the <host identifier>. For every such host language data type an equivalent SQL <data type> is specified in Subclause 21.3, "<embedded SQL Ada program>", Subclause 21.4, "<embedded SQL C program>", Subclause 21.5, "<embedded SQL COBOL program>", Subclause 21.6, "<embedded SQL Fortran program>", Subclause 21.7, "<embedded SQL MUMPS program>", Subclause 21.8, "<embedded SQL Pascal program>", and Subclause 21.9, "<embedded SQL PL/I program>".

14) If one or more <host variable definition>s that specify SQLSTATE or SQLCODE appear in an <embedded SQL host program>, then the <host variable definition>s shall be such that a host language reference to SQLSTATE or SQLCODE, respectively, is valid at every <embedded SQL statement>, including <embedded SQL statement>s that appear in any subprograms contained in that <embedded SQL host program>. No <embedded SQL statement> shall precede any of its applicable status code definitions in the text of the main program or subprograms that comprise the <embedded SQL host program>.

15) Given an <embedded SQL host program> *H*, there is an implied standard-conforming SQL <module> *M* and an implied standard-conforming host program *P* derived from *H*. The derivation of the implied program *P* and the implied <module> *M* of an <embedded SQL host program> *H* effectively precedes the processing of any host language program text manipulation commands such as inclusion or copying of text.

Given an <embedded SQL host program> *H* with an implied <module> *M* and an implied program *P* defined as above:

a) The implied <module> *M* of *H* shall be a standard-conforming SQL <module>.

b) If *H* is an <embedded SQL Ada program>, an <embedded SQL C program>, an <embedded SQL COBOL program>, an <embedded SQL Fortran program>, an <embedded SQL MUMPS program>, an <embedded SQL Pascal program>, or an <embedded SQL PL/I program>, then the implied program *P* shall be a standard-conforming Ada program, a standard-conforming C program, a standard-conforming COBOL program, a standard-conforming Fortran program, a standard-conforming MUMPS program, a standard-conforming Pascal program, or standard-conforming PL/I program, respectively.

16) *M* is derived from *H* as follows:

a) *M* contains a <module name clause> whose <module name> is either implementation-dependent or is omitted.

b) *M* contains a <module character set specification> that is identical to the explicit or implicit <embedded character set declaration> with the keyword ''SQL'' removed.

c) *M* contains a <language clause> that specifies either ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, where *H* is respectively an <embedded SQL Ada program>, an <embedded SQL C program>, an <embedded SQL COBOL program>, an <embedded SQL Fortran program>, an <embedded SQL MUMPS program>, an <embedded SQL Pascal program>, or an <embedded SQL PL/I program>.

d) *M* contains a <module authorization clause> that specifies SCHEMA <schema name>, where the value of <schema name> is implementation-dependent.

e) For every <declare cursor> *EC* contained in *H*, *M* contains one <declare cursor> *PC* and one <routine> *PS* that contains an <open statement> that references *PC*. The <routine header> of *PS* specifies ''PROCEDURE''. The <procedure name> of *PS* is implementation-dependent. *PC* is a copy of *EC* in which each distinct <embedded variable name> has been replaced with a distinct implementation-dependent
[ANSI] <host parameter name>.
[ISO] <parameter name>.
PS contains a <parameter declaration> for each
[ANSI] <host parameter name>
[ISO] <parameter name>
contained in *PC*. The <parameter declaration> that corresponds to a given <embedded variable name> *V* that is contained in *EC* specifies the
[ANSI] <host parameter name>
[ISO] <parameter name>
with which *V* was replaced, and the SQL data type that corresponds to the host language data type of *V* as specified in Subclause 13.4, "Data type correspondences". If *H* contains a <host variable definition> that specifies SQLSTATE or SQLCODE, then *PS* contains a <parameter declaration> that specifies SQLSTATE or SQLCODE, respectively. If *H* contains neither a <host variable definition> that specifies SQLSTATE nor a <host variable definition> that specifies SQLCODE, then *PS* contains a <parameter declaration> that specifies SQLCODE. The order of <parameter declaration>s in *PS* is implementation-dependent.

f) For every <dynamic declare cursor> *EC* in *H*, *M* contains one <dynamic declare cursor> *PC* that is a copy of *EC*.

g) *M* contains one <temporary table declaration> for each <temporary table declaration> contained in *H*. Each <temporary table declaration> of *M* is a copy of the corresponding <temporary table declaration> of *H*.

h) *M* contains one <temporary view definition> for each <temporary view definition> contained in *H*. Each <temporary view definition> of *M* is a copy of the corresponding <temporary view definition> of *H*.

ANSI Only–SQL3

i) *M* contains one <SQL variable declaration> for each <SQL variable declaration> contained in *H*. Each <SQL variable declaration> of *M* is a copy of the corresponding <SQL variable declaration> of *H*.

j) *M* contains a <routine> for each <SQL procedure statement> contained in *H*. The <routine> *PS* of *M* corresponding with an <SQL procedure statement> *ES* of *H* is defined as follows:

Case:

i) If *ES* is not an <open statement>, then:

1) The <routine header> of *PS* specifies ''PROCEDURE''.

2) The <procedure name> of *PS* is implementation-dependent.

3) The <SQL procedure statement> of *PS* is a copy of *ES* in which each distinct <embedded variable name> has been replaced with the same distinct implementation-dependent
ANSI <host parameter name>.
ISO <parameter name>.

4) *PS* contains a <parameter declaration> for each distinct
ANSI <host parameter name>
ISO <parameter name>
contained in the <SQL procedure statement> of *PS*.

5) The <parameter declaration> corresponding to a given <embedded variable name> *V* that is contained in *ES* specifies the
ANSI <host parameter name>
ISO <parameter name>
with which *V* was replaced and the SQL <data type> that corresponds to the host language data type of *V*.

6) Whether one <routine> of *M* can correspond to more than one <SQL procedure statement> of *H* is implementation-dependent.

7) If *H* contains a <host variable definition> that specifies SQLSTATE or SQLCODE, then *PS* contains a <parameter declaration> that specifies SQLSTATE or SQLCODE, respectively. If *H* contains neither a <host variable definition> that specifies SQLSTATE nor a <host variable definition> that specifies SQLCODE, then *PS* contains a <parameter declaration> that specifies SQLCODE.

8) The order of the <parameter declaration>s is implementation-dependent.

ii) If *ES* is an <open statement>, then:

1) Let *EC* be the <declare cursor> in *H* referenced by *ES*.

2) *PS* is the <routine> in *M* that contains an <open statement> that references the <declare cursor> in M corresponding to *EC*.

k) *M* contains one <external function declaration> for each <external function declaration> contained in *H*. Each <external function declaration> of *M* is a copy of the corresponding <external function declaration> of *H*.

17) *P* is derived from *H* as follows:

a) Each <embedded SQL begin declare>, <embedded SQL end declare>, and <embedded character set declaration> has been deleted. If the embedded host language is MUMPS, then each <embedded SQL MUMPS declare> has been deleted.

b) Each <host variable definition> in an <embedded SQL declare section> has been replaced by a valid data definition in the target host language according to the Syntax Rules specified in an <embedded SQL Ada program>, <embedded SQL C program>, <embedded SQL COBOL program>, <embedded SQL Fortran program>, <embedded SQL Pascal program>, or an <embedded SQL PL/I program> clause.

c) Each <embedded SQL statement> that contains a <declare cursor>, a <dynamic declare cursor>,
ANSI an <SQL variable declaration>,
an <external function declaration>, a <temporary view definition>, or a <temporary table declaration> has been deleted, and every <embedded SQL statement> that contains an <embedded exception declaration> has been replaced with statements of the host language that will have the effect specified by the General Rules of Subclause 21.2, "<embedded exception declaration>".

d) Each <embedded SQL statement> that contains an <SQL procedure statement> has been replaced by host language statements that perform the following actions:

i) A host language procedure or subroutine call of the <routine> of the implied <module> *M* of *H* that corresponds with the <SQL procedure statement>.

If the <SQL procedure statement> is not an <open statement>, then the arguments of the call include each distinct <host identifier> contained in the <SQL procedure statement> together with the SQLCODE <host identifier>, or the SQLSTATE <host identifier>, or both. If *H* contains neither a <host variable definition> that specifies SQLSTATE nor a <host variable definition> that specifies SQLCODE, then the arguments of the call include SQLCODE. If the <SQL procedure statement> is an <open statement>, then the arguments of the call include each distinct <host identifier> contained in the corresponding <declare cursor> of *H* together with the SQLCODE <host identifier>, or the SQLSTATE <host identifier>, or both. If *H* contains neither a <host variable definition> that specifies SQLSTATE nor a <host variable definition> that specifies SQLCODE, then the arguments of the call include SQLCODE.

The order of the arguments in the call corresponds with the order of the corresponding <parameter declaration>s in the corresponding <routine>.

Note: In an <embedded SQL Fortran program>, the ''SQLCODE'' variable is abbreviated to ''SQLCOD'' and the ''SQLSTATE'' variable may be abbreviated to ''SQLSTA''. See the Syntax Rules of Subclause 21.6, "<embedded SQL Fortran program>".

ii) Exception actions, as specified in Subclause 21.2, "<embedded exception declaration>".

****Editor's Note****

The preceding Rule (nor any other place in this document) does not describe how those constructs that are not standard-conforming host language constructs are to be reconstructed to be conforming. This has have been identified as a Possible Problem. See Possible Problems **093** and **124** in the Editor's Notes.

Note: SQLSTATE is the preferred status parameter. The SQLCODE status parameter is a deprecated feature that is supported for compatibility with earlier versions of this
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) The interpretation of an <embedded SQL host program> *H* is defined to be equivalent to the interpretation of the implied program *P* of *H* and the implied <module> *M* of *H*.

2) If the cursor mode of the current SQL-transaction is *cascade off*, and the <embedded SQL statement> is other than a <close statement>, a <delete statement: positioned>, a <fetch statement>, an <open statement>, an <update statement: positioned>, a <dynamic close statement>, a <dynamic delete statement: positioned>, a <dynamic fetch statement>, a <dynamic open statement>, or a <dynamic update statement: positioned>, then an exception condition is raised: *invalid SQL statement*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <statement or declaration> shall not specify a <temporary view definition>,
   ANSI an <SQL variable declaration>,
   or a <routine> that specifies an external routine.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) An <embedded SQL declare section> shall not contain an <embedded character set declaration>.

   b) No two <host variable definition>s shall specify the same variable name.

21.2 <embedded exception declaration>

Function

Specify the action to be taken when an SQL-statement causes a specific class of condition to be raised.

Format

```
<embedded exception declaration> ::=
       WHENEVER <condition> <condition action>

<condition> ::=
       | <old condition>
       | <SQL condition>

<old condition> ::= SQLERROR

<SQL condition> ::=
         <major category>
       | SQLSTATE ( <SQLSTATE class value> [ , <SQLSTATE subclass value> ] )
       | CONSTRAINT <constraint name>

<major category> ::=
         SQLEXCEPTION
       | SQLWARNING
       | NOT FOUND

<SQLSTATE class value> ::=
       <SQLSTATE char><SQLSTATE char> !! See the Syntax Rules.

<SQLSTATE subclass value> ::=
       <SQLSTATE char><SQLSTATE char><SQLSTATE char> !! See the Syntax Rules.

<SQLSTATE char> ::= <simple Latin upper case letter> | <digit>

<condition action> ::=
       CONTINUE | <go to>

<go to> ::=
       { GOTO | GO TO } <goto target>

<goto target> ::=
         <host label identifier>
       | <unsigned integer>
       | <host PL/I label variable>

<host label identifier> ::= !! See the Syntax Rules.

<host PL/I label variable> ::= !! See the Syntax Rules.
```

Syntax Rules

1) If an <embedded exception declaration> that specifies <old condition> is contained in the <embedded SQL host program>, then no <embedded exception declaration>s that specifies <SQL condition> shall exist in the same <embedded SQL host program>.

2) SQLWARNING, NOT FOUND, and SQLEXCEPTION correspond to SQLSTATE class values corresponding to categories W, N, and X in Table 30, "SQLSTATE class and subclass values", respectively.

3) An <embedded exception declaration> contained in an <embedded SQL host program> applies to an <SQL procedure statement> contained in that <embedded SQL host program> if and only if the <SQL procedure statement> appears after the <embedded exception declaration> that has condition *C* in the text sequence of the <embedded SQL host program> and no other <embedded exception declaration> *E* that satisfies one of the following conditions appears between the <embedded exception declaration> and the <SQL procedure statement> in the text sequence of the <embedded SQL host program>.

   Let *D* be the <condition> contained in *E*.

   Case:

   a) *D* is the same as *C*.

   b) *D* is a <major catagory> and belongs to the same class to which *C* belongs.

   c) *D* contains an <SQLSTATE class value>, but does not contain an <SQLstate subclass value>, and *E* contains the same <SQLstate class value> that *C* contains.

   d) *D* contains the <SQLSTATE class value> that corresponds to *integrity constraint violation* and *C* contains CONSTRAINT.

4) In the values of <SQLSTATE class value> and <SQLSTATE subclass value>, there shall be no <separator> between the <SQLSTATE char>s.

5) The values of <SQLSTATE class value> and <SQLSTATE subclass value> shall correspond to class values and subclass values, respectively, specified in Table 30, "SQLSTATE class and subclass values".

6) If an <embedded exception declaration> specifies a <go to>, then the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> shall be such that a host language GO TO statement specifying that <host label identifier>, <host PL/I label variable>, or <unsigned integer> is valid at every <SQL procedure statement> to which the <embedded exception declaration> applies.

   Note:

   1) If an <embedded exception declaration> is contained in an <embedded SQL Ada program>, then the <goto target> of a <go to> should specify a <host label identifier> that is a label_name in the containing <embedded SQL Ada program>.

   2) If an <embedded exception declaration> is contained in an <embedded SQL C program>, then the <goto target> of a <go to> should specify a <host label identifier> that is a label in the containing <embedded SQL C program>.

   3) If an <embedded exception declaration> is contained in an <embedded SQL COBOL program>, then the <goto target> of a <go to> should specify a <host label identifier> that is a section-name or an unqualified paragraph-name in the containing <embedded SQL COBOL program>.

4) If an <embedded exception declaration> is contained in an <embedded SQL Fortran program>, then the <goto target> of a <go to> should be an <unsigned integer> that is the statement label of an executable statement that appears in the same program unit as the <go to>.

5) If an <embedded exception declaration> is contained in an <embedded SQL MUMPS program>, then the <goto target> of a <go to> should be a gotoargument that is the statement label of an executable statement that appears in the same <embedded SQL MUMPS program>.

6) If an <embedded exception declaration> is contained in an <embedded SQL Pascal program>, then the <goto target> of a <go to> should be an <unsigned integer> that is a label.

7) If an <embedded exception declaration> is contained in an <embedded SQL PL/I program>, then the <goto target> of a <go to> should specify either a <host label identifier> or a <host PL/I label variable>.

Case:

a) If <host label identifier> is specified, then the <host label identifier> should be a label constant in the containing <embedded SQL PL/I program>.

b) If <host PL/I label variable> is specified, then the <host PL/I label variable> should be a PL/I label variable declared in the containing <embedded SQL PL/I program>.

Access Rules

None.

General Rules

1) Immediately after the execution of an <SQL procedure statement> *STMT* in an <embedded SQL host program> that returns an SQLSTATE value other than *successful completion*:

a) Let *E* be the set of <embedded exception declaration>s that are contained in the <embedded SQL host program> containing *STMT*, that applies to *STMT*, and that specifies an <condition action> that is <go to>.

b) Let *CV* and *SCV* be respectively the values of the class and subclass of the SQLSTATE value that indicates the result of the <SQL procedure statement>.

c) If the execution of the <SQL procedure statement> caused the violation of one or more constraints or assertions, then:

i) Let *ECN* be the set of <embedded exception declaration>s in *E* that specify CONSTRAINT and the <constraint name> of a constraint that was violated by execution of *STMT*.

ii) If *ECN* contains more than one <embedded exception declaration>, then an implementation-dependent <embedded exception declaration> is chosen from *ECN*; otherwise, the single <embedded exception declaration> in *ECN* is chosen.

iii) A GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> chosen from *ECN*.

d) Otherwise:

i) Let *ECS* be the set of <embedded exception declaration>s in *E* that specify SQLSTATE, an <SQLSTATE class value>, and an <SQLSTATE subclass value>.

ii) If *ECS* contains an <embedded exception declaration> *EY* that specifies an <SQLSTATE class value> identical to *CV* and an <SQLSTATE subclass value> identical to *SCV*, then a GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> *EY*.

iii) Otherwise:

1) Let *EC* be the set of <embedded exception declaration>s in *E* that specify SQLSTATE and an <SQLSTATE class value> without an <SQLSTATE subclass value>.

2) If *EC* contains an <embedded exception declaration> *EY* that specifies an <SQLSTATE class value> identical to *CV*, then a GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> *EY*.

3) Otherwise:

A) Let *EE* be the set of <embedded exception declaration>s in *E* that specify SQLERROR.

B) If *EE* contains an <embedded exception declaration> *EY* and *CV* belongs to Category W or Category X in Table 30, "SQLSTATE class and subclass values", then a GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> *EY*.

C) Otherwise:

I) Let *EX* be the set of <embedded exception declaration>s in *E* that specify SQLEXCEPTION.

II) If *EX* contains an <embedded exception declaration> *EY* and *CV* belongs to Category X in Table 30, "SQLSTATE class and subclass values", then a GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> *EY*.

III) Otherwise:

1) Let *EW* be the set of <embedded exception declaration>s in *E* that specify SQLWARNING.

2) If *EW* contains an <embedded exception declaration> *EY* and *CV* belongs to Category W in Table 30, "SQLSTATE class and subclass values", then a GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> *EY*.

3) Otherwise, let *ENF* be the set of <embedded exception declaration>s in *E* that specify NOT FOUND. If *ENF* contains an <embedded exception declaration> *EY* and *CV* belongs to Category N in Table 30, "SQLSTATE class and subclass values", then a GO TO statement of the host language

is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> *EY*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <SQL condition> shall not specify SQLSTATE or CONSTRAINT.

   b) A <major category> shall not specify SQLEXCEPTION or SQLWARNING.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

21.3 <embedded SQL Ada program>

Function

Specify an <embedded SQL Ada program>.

Format

```
<embedded SQL Ada program> ::= !! See the Syntax Rules.

<Ada variable definition> ::=
       <Ada host identifier> [ { <comma> <Ada host identifier> }... ] :
       <Ada type specification> [ <Ada initial value> ]

<Ada initial value> ::=
       <Ada assignment operator> <character representation>...

<Ada assignment operator> ::= <colon><equals operator>

<Ada host identifier> ::= !! See the Syntax Rules.

<Ada type specification> ::=
         <Ada qualified type specification>
       | <Ada unqualified type specification>

<Ada qualified type specification> ::=
         SQL_STANDARD.CHAR [ CHARACTER SET [ IS ] <character set specification> ]
             <left paren> 1 <double period> <length> <right paren>
       | SQL_STANDARD.BIT <left paren> 1 <double period> <length> <right paren>
       | SQL_STANDARD.SMALLINT
       | SQL_STANDARD.INT
       | SQL_STANDARD.REAL
       | SQL_STANDARD.DOUBLE_PRECISION
       | SQL_STANDARD.BOOLEAN
       | SQL_STANDARD.SQLCODE_TYPE
       | SQL_STANDARD.SQLSTATE_TYPE
       | SQL_STANDARD.INDICATOR_TYPE

<Ada unqualified type specification> ::=
         CHAR <left paren> 1 <double period> <length> <right paren>
       | BIT <left paren> 1 <double period> <length> <right paren>
       | SMALLINT
       | INT
       | REAL
       | DOUBLE_PRECISION
       | BOOLEAN
       | SQLCODE_TYPE
       | SQLSTATE_TYPE
       | INDICATOR_TYPE
```

Syntax Rules

1) An <embedded SQL Ada program> is a compilation unit that consists of Ada text and SQL text. The Ada text shall conform to the Ada standard
ANSI ANSI/MIL-STD-1815A.
ISO ISO/IEC 8652.
The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever an Ada statement may be specified. An <embedded SQL statement> may be prefixed by an Ada label.

3) An <Ada host identifier> is any valid Ada identifier. An <Ada host identifier> shall be contained in an <embedded SQL Ada program>.

4) An <Ada variable definition> defines one or more host variables.

5) An <Ada variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL Ada program> (see the Syntax Rules of Subclause 21.1, "<embedded SQL host program>"):

   a) Any optional CHARACTER SET specification shall be removed from an <Ada qualified type specification>.

   b) The <length> specified in a CHAR declaration of any <Ada qualified type specification> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <Ada host identifier> specified in the containing <Ada variable definition>.

   The modified <Ada variable definition> shall be a valid Ada object-declaration in the program derived from the <embedded SQL Ada program>.

6) An <Ada variable definition> shall be specified within the scope of Ada **with** and **use** clauses that specify the following:

   with SQL_STANDARD;

   use SQL_STANDARD;

   use SQL_STANDARD.CHARACTER_SET;

7) The <character representation> sequence in an <Ada initial value> specifies an initial value to be assigned to the Ada variable. It shall be a valid Ada specification of an initial value.

8) CHAR describes a character string variable whose equivalent SQL data type is CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

9) BIT describes a bit string variable. The equivalent SQL data type is BIT with the same length.

10) INT and SMALLINT describe exact numeric variables. The equivalent SQL data types are INTEGER and SMALLINT, respectively.

11) REAL and DOUBLE_PRECISION describe approximate numeric variables. The equivalent SQL data types are REAL and DOUBLE PRECISION, respectively.

12) BOOLEAN describes a boolean variable. The equivalent SQL data type is BOOLEAN.

13) SQLCODE_TYPE describes an exact numeric variable whose precision is the implementation-defined precision defined for the SQLCODE parameter. SQLSTATE_TYPE describes a character string variable whose length is the length of the SQLSTATE parameter, five characters.

14) INDICATOR_TYPE describes an exact numeric variable whose specific data type is any <exact numeric type> with a scale of 0.

15) If an <embedded SQL Ada program> contains neither an <Ada variable definition> that specifies SQLSTATE and that is defined with an <Ada type specification> that specifies SQL_STANDARD.CHAR or CHAR with <length> 5, nor an <Ada variable definition> that specifies SQLCODE and that is defined with an <Ada type specification> that specifies SQL_STANDARD.INTEGER or INTEGER, then it is assumed that the <embedded SQL Ada program> contains a variable named SQLCODE defined with a data type of INTEGER.

Note: SQLSTATE is the preferred status parameter. The SQLCODE status parameter is a deprecated feature that is supported for compatibility with earlier versions of this
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 21.1, "<embedded SQL host program>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) An <Ada variable definition> shall not specify a bit string variable.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) An <Ada qualified type specification> shall not contain a <character set specification>.

21.4 <embedded SQL C program>

Function

Specify an <embedded SQL C program>.

Format

```
<embedded SQL C program> ::= !! See the Syntax Rules.

<C variable definition> ::=
        [ <C storage class> ]
        [ <C class modifier> ]
        <C variable specification>
      <semicolon>

<C variable specification> ::=
        <C numeric variable>
      | <C character variable>
      | <C derived variable>

<C storage class> ::=
        auto
      | extern
      | static

<C class modifier> ::= const | volatile

<C numeric variable> ::=
      { long | short | float | double }
        <C host identifier> [ <C initial value> ]
            [ { <comma> <C host identifier> [ <C initial value> ] }... ]

<C character variable> ::=
      char [ CHARACTER SET [ IS ] <character set specification> ]
        <C host identifier> <C array specification> [ <C initial value> ]
          [ { <comma> <C host identifier> <C array specification>
                  [ <C initial value> ] }... ]

<C array specification> ::=
      <left bracket> <length> <right bracket>

<C host identifier> ::= !! See the Syntax Rules.

<C derived variable> ::=
        <C VARCHAR variable>
      | <C bit variable>

<C VARCHAR variable> ::=
      VARCHAR [ CHARACTER SET [ IS ] <character set specification> ]
          <C host identifier> <C array specification>
            [ { <comma> <C host identifier> <C array specification> }... ]

<C bit variable> ::=
      BIT <C host identifier> <C array specification> [ <C initial value> ]
        [ { <comma> <C host identifier> <C array specification>
                  [ <C initial value> ] }... ]

<C initial value> ::=
```

```
<equals operator> <character representation>...
```

Syntax Rules

1) An <embedded SQL C program> is a compilation unit that consists of C text and SQL text. The C text shall conform to to the C standard
   ANSI ANSI X3.159.
   ISO ISO/IEC 9899.
   The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever a C statement may be specified within a function block. If the C statement could include a label prefix, then the <embedded SQL statement> may be immediately preceded by a label prefix.

3) A <C host identifier> is any valid C variable identifier. A <C host identifier> shall be contained in an <embedded SQL C program>.

4) A <C variable definition> defines one or more host variables.

5) A <C variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL C program> (see the Syntax Rules of Subclause 21.1, "<embedded SQL host program>"):

   a) Any optional CHARACTER SET specification shall be removed from a <C VARCHAR variable> or a <C character variable>.

   b) The syntax ''VARCHAR'' shall be replaced by ''char'' in any <C VARCHAR variable>.

   c) The syntax ''BIT'' shall be replaced by ''char'' in any <C bit variable>.

   d) The <length> specified in a <C array specification> in any <C bit variable> shall be replaced by a length equal to the smallest integer not less than *L*/*B*, as defined in the Syntax Rules of this Subclause.

   e) The <length> specified in a <C array specification> in any <C character variable> or in any <C VARCHAR variable> that contained a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <C host identifier> specified in the containing <C variable definition>.

   The modified <C variable definition> shall be a valid C data declaration in the program derived from the <embedded SQL C program>.

6) The <character representation> sequence contained in a <C initial value> specifies an initial value to be assigned to the C variable. It shall be a valid C specification of an initial value.

7) Except for array specifications for character strings and bit strings, a <C variable definition> shall specify a scalar type.

8) In a <C variable definition>, the words ''VARCHAR'', ''CHARACTER'', ''SET'', ''IS'', ''BIT'', and ''VARYING'' may be specified in any combination of upper case and lower case letters (see the Syntax Rules of Subclause 5.2, "<token> and <separator>").

9) In a <C character variable> or a <C VARCHAR variable>, if a <character set specification> is specified, then the equivalent SQL data type is CHARACTER or CHARACTER VARYING whose character repertoire is the same as the repertoire specified by the <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

10) Each <C host identifier> specified in a <C character variable> describes a fixed-length character string. The length is specified by the <length> of the <C array specification>. The value in the host variable is terminated by a null character and the position occupied by this null character is included in the length of the host variable. The equivalent SQL data type is CHARACTER whose length is one less than the <length> of the <C array specification> and whose value does not include the terminating null character. The <length> shall be greater than 1.

11) Each <C host identifier> specified in a <C VARCHAR variable> describes a variable-length character string. The maximum length is specified by the <length> of the <C array specification>. The value in the host variable is terminated by a null character and the position occupied by this null character is included in the maximum length of the host variable. The equivalent SQL data type is CHARACTER VARYING whose maximum length is 1 less than the <length> of the <C array specification> and whose value does not include the terminating null character. The <length> shall be greater than 1.

12) Each <C host identifier> specified in a <C bit variable> describes a fixed-length bit string. The value in the host variable has a BIT_LENGTH of <length>. Let *B* be the number of bits in a C char and let *L* be the <length> of the <C array specification>. The length of an equivalent C char variable is the smallest integer that is not less than the result of *L*/*B*. The equivalent SQL data type is BIT whose length is *L*.

13) ''long'' describes an exact numeric variable. The equivalent SQL data type is INTEGER or BOOLEAN.

14) ''short'' describes an exact numeric variable. The equivalent SQL data type is SMALLINT.

15) ''float'' describes an approximate numeric variable. The equivalent SQL data type is REAL.

16) ''double'' describes an approximate numeric variable. The equivalent SQL data type is DOUBLE PRECISION.

17) If an <embedded SQL C program> contains neither a <C variable definition> that specifies SQLSTATE and that is defined with a <C character variable> that specifies ''char'' with a <C array specification> that is 6, nor a <C variable definition> that specifies SQLCODE and that is defined with a <C numeric variable> that specifies ''long'', then it is assumed that the <embedded SQL C program> contains a variable named SQLCODE defined with a data type of ''long''.

Note: SQLSTATE is the preferred status parameter. The SQLCODE status parameter is a deprecated feature that is supported for compatibility with earlier versions of this
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 21.1, "<embedded SQL host program>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <C derived variable> shall not be a <C bit variable>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <C derived variable> shall not be a <C VARCHAR variable>.

   b) A <C variable definition> shall not contain a <character set specification>.

21.5 <embedded SQL COBOL program>

Function

Specify an <embedded SQL COBOL program>.

Format

```
<embedded SQL COBOL program> ::= !! See the Syntax Rules.

<COBOL variable definition> ::=
       {01|77} <COBOL host identifier> <COBOL type specification>
         [ <character representation>... ] <period>

<COBOL host identifier> ::= !! See the Syntax Rules.

<COBOL type specification> ::=
         <COBOL character type>
       | <COBOL bit type>
       | <COBOL numeric type>
       | <COBOL integer type>

<COBOL character type> ::=
       [ CHARACTER SET [ IS ] <character set specification> ]
       { PIC | PICTURE } [ IS ] { X [ <left paren> <length> <right paren> ] }...

<COBOL bit type> ::=
       { PIC | PICTURE } [ IS ] { X [ <left paren> <length> <right paren> ] }...
         USAGE [ IS ] BIT

<COBOL numeric type> ::=
       { PIC | PICTURE } [ IS ]
         S <COBOL nines specification>
       [ USAGE [ IS ] ] DISPLAY SIGN LEADING SEPARATE

<COBOL nines specification> ::=
         <COBOL nines> [ V [ <COBOL nines> ] ]
       | V <COBOL nines>

<COBOL integer type> ::=
         <COBOL computational integer>
       | <COBOL binary integer>

<COBOL computational integer> ::=
       { PIC | PICTURE } [ IS ] S<COBOL nines>
         [ USAGE [ IS ] ] { COMP | COMPUTATIONAL } ]

<COBOL binary integer> ::=
       { PIC | PICTURE } [ IS ] S<COBOL nines>
         [ USAGE [ IS ] ] BINARY

<COBOL nines> ::= { 9 [ <left paren> <length> <right paren> ] }...
```

Syntax Rules

1) An <embedded SQL COBOL program> is a compilation unit that consists of COBOL text and SQL text. The COBOL text shall conform to to the COBOL standard
ANSI ANSI X3.23.
ISO ISO 1989.
The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> in an <embedded SQL COBOL program> may be specified wherever a COBOL statement may be specified in the Procedure Division of the <embedded SQL COBOL program>. If the COBOL statement could be immediately preceded by a paragraph-name, then the <embedded SQL statement> may be immediately preceded by a paragraph-name.

3) A <COBOL host identifier> is any valid COBOL data-name. A <COBOL host identifier> shall be contained in an <embedded SQL COBOL program>.

4) A <COBOL variable definition> is a restricted form of COBOL data description entry that defines a host variable.

5) A <COBOL variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL COBOL program> (see the Syntax Rules of Subclause 21.1, "<embedded SQL host program>".

   a) Any optional CHARACTER SET specification shall be removed from a <COBOL character type>.

   b) The syntax ''USAGE IS BIT'' shall be deleted.

   c) The <length> specified in any <COBOL bit type> shall be replaced by a length equal to the smallest integer not less than *L*/*B*, as defined in the Syntax Rules of this Subclause.

   d) The <length> specified in any <COBOL character type> that contained a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <COBOL host identifier> specified in the containing <COBOL variable definition>.

   The modified <COBOL variable definition> shall be a valid data description entry in the Data Division of the program derived from the <embedded SQL COBOL program>.

6) The optional <character representation> sequence in a <COBOL variable definition> may specify a VALUE clause. Whether other clauses may be specified is implementation-defined. The <character representation> sequence shall be such that the <COBOL variable definition> is a valid COBOL data description entry.

7) A <COBOL character type> describes a character string variable whose equivalent SQL data type is CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

8) A <COBOL bit type> describes a bit string variable. Let *B* be the number of bits in a COBOL character and let *L* be the <length> of the <COBOL bit type>. The length of an equivalent COBOL character variable is the smallest integer not less than *L*/*B*. The equivalent SQL data type is BIT whose length is the <length> of the <COBOL bit type>. If the length of the <COBOL bit type> is 1, then the equivalent SQL data type may be BOOLEAN.

9) A <COBOL numeric type> describes an exact numeric variable. The equivalent SQL data type is NUMERIC of the same precision and scale.

10) A <COBOL computational integer> describes an exact numeric variable.

Note: This <COBOL type specification> is supported only for SQLCODE for compatibility with earlier versions of this
ANSI American
ISO International
Standard. The SQLCODE status parameter is a deprecated feature that is supported for compatibility with earlier versions of this
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

11) A <COBOL binary integer> describes an exact numeric variable. The equivalent SQL data type is SMALLINT or INTEGER.

12) If an <embedded SQL COBOL program> contains neither a <COBOL variable definition> that specifies SQLSTATE and that is defined with a <COBOL character type> that specifies

{01|77} SQLSTATE PICTURE X(5)

nor a <COBOL variable definition> that specifies SQLCODE and that is defined with a <COBOL type specification> that specifies

{01|77} SQLCODE PICTURE S9(*PC*) USAGE COMP

where *PC* is the implementation-defined precision specified for a COBOL SQLCODE parameter in Subclause 10.3, "<routine invocation>", then it is assumed <embedded SQL COBOL program> contains a variable named SQLCODE defined with a data type of

{01|77} SQLCODE PICTURE S9(*PC*) USAGE COMP

where *PC* is the implementation-defined precision specified for a COBOL SQLCODE parameter in Subclause 10.3, "<routine invocation>".

Note: SQLSTATE is the preferred status parameter. The SQLCODE status parameter is a deprecated feature that is supported for compatibility with earlier versions of this
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 21.1, "<embedded SQL host program>".

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) A <COBOL type specification> shall not be a <COBOL bit type>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <COBOL character type> shall not contain a <character set specification>.

21.6 <embedded SQL Fortran program>

Function

Specify an <embedded SQL Fortran program>.

Format

```
<embedded SQL Fortran program> ::= !! See the Syntax Rules.

<Fortran variable definition> ::=
       <Fortran type specification>
       <Fortran host identifier> [ { <comma> <Fortran host identifier> }... ]

<Fortran host identifier> ::= !! See the Syntax Rules.

<Fortran type specification> ::=
         CHARACTER [ <asterisk> <length> ]
             [ CHARACTER SET [ IS ] <character set specification> ]
       | BIT [ <asterisk> <length> ]
       | INTEGER
       | REAL
       | DOUBLE PRECISION
       | LOGICAL
```

Syntax Rules

1) An <embedded SQL Fortran program> is a compilation unit that consists of Fortran text and SQL text. The Fortran text shall conform to to the Fortran
   ANSI standards ANSI X3.9 and ANSI X3.198.
   ISO standard ISO 1539.
   The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever an executable Fortran statement may be specified. An <embedded SQL statement> that precedes any executable Fortran statement in the containing <embedded SQL Fortran program> shall not have a Fortran statement number. Otherwise, if the Fortran statement could have a statement number then the <embedded SQL statement> can have a statement number.

3) Blanks are significant in <embedded SQL statement>s. The rules for <separator>s in an <embedded SQL statement> are as specified in Subclause 5.2, "<token> and <separator>".

4) A <Fortran host identifier> is any valid Fortran variable name with all <space> characters removed. A <Fortran host identifier> shall be contained in an <embedded SQL Fortran program>.

5) A <Fortran variable definition> is a restricted form of Fortran type-statement that defines one or more host variables.

6) A <Fortran variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL Fortran program> (see the Syntax Rules Subclause 21.1, "<embedded SQL host program>".

   a) Any optional CHARACTER SET specification shall be removed from the CHARACTER alternative in a <Fortran type specification>.

   b) The <length> specified in the CHARACTER alternative of any <Fortran type specification> that contained a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <Fortran host identifier> specified in the containing <Fortran variable definition>.

   c) The syntax ''BIT'' shall be replaced by ''CHARACTER'' in any BIT alternative of a <Fortran type specification>.

   d) The <length> specified in any BIT alternative of a <Fortran type specification> shall be replaced by a length equal to the smallest integer not less than *L*/*B*, as defined in the Syntax Rules of this Subclause.

   The modified <Fortran variable definition> shall be a valid Fortran type-statement in the program derived from the <embedded SQL Fortran program>.

7) CHARACTER describes a character string variable whose equivalent SQL data type is CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

8) BIT describes a bit string variable. Let *B* be the number of bits in a Fortran character and let *L* be the <length> of the bit string variable. The length of an equivalent Fortran character variable is the smallest integer not less than *L*/*B*. The equivalent SQL data type is BIT whose length is the <length> of the bit string variable.

9) INTEGER describes an exact numeric variable. The equivalent SQL data type is INTEGER.

10) REAL describes an approximate numeric variable. The equivalent SQL data type is REAL.

11) DOUBLE PRECISION describes an approximate numeric variable. The equivalent SQL data type is DOUBLE PRECISION.

12) LOGICAL describes a boolean variable. The equivalent SQL data type is BOOLEAN.

13) If an <embedded SQL Fortran program> contains neither a <Fortran variable definition> that specifies SQLSTATE or SQLSTA and that is defined with a <Fortran type specification> that specifies CHARACTER with <length> 5, nor a <Fortran variable definition> that specifies SQLCOD and that is defined with a <Fortran type specification> that specifies INTEGER, then it is assumed that the <embedded SQL Fortran program> contains a variable named SQLCOD defined with a data type of INTEGER.

   Note: SQLSTATE (which may be abbreviated ''SQLSTA'') is the preferred status parameter. The SQLCODE (SQLCOD) status parameter is a deprecated feature that is supported for compatibility with earlier versions of this
   ANSI American
   ISO International
   Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 21.1, "<embedded SQL host program>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <Fortran type specification> shall not specify BIT.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <Fortran type specification> shall not contain a <character set specification>.

21.7 <embedded SQL MUMPS program>

Function

Specify an <embedded SQL MUMPS program>.

Format

```
<embedded SQL MUMPS program> ::= !! See the Syntax Rules.

<MUMPS variable definition> ::=
       { <MUMPS numeric variable> | <MUMPS character variable> } <semicolon>

<MUMPS character variable> ::=
       VARCHAR <MUMPS host identifier> <MUMPS length specification>
         [ { <comma> <MUMPS host identifier> <MUMPS length specification> }... ]

<MUMPS host identifier> ::= !! See the Syntax Rules.

<MUMPS length specification> ::=
       <left paren> <length> <right paren>

<MUMPS numeric variable> ::=
       <MUMPS type specification>
         <MUMPS host identifier> [ { <comma> <MUMPS host identifier> }... ]

<MUMPS type specification> ::=
         INT
       | DEC [ <left paren> <precision> [ <comma> <scale> ] <right paren> ]
       | REAL
```

Syntax Rules

1) An <embedded SQL MUMPS program> is a compilation unit that consists of MUMPS text and SQL text. The MUMPS text shall conform to to the MUMPS standard
   ANSI ANSI/MDC X11.1.
   ISO ISO/IEC 11756.
   The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) A <MUMPS host identifier> is any valid MUMPS variable name. A <MUMPS host identifier> shall be contained in an <embedded SQL MUMPS program>.

3) An <embedded SQL statement> may be specified wherever a MUMPS command may be specified.

4) A <MUMPS variable definition> defines one or more host variables.

5) The <MUMPS character variable> describes a variable-length character string. The equivalent SQL data type is CHARACTER VARYING whose maximum length is the <length> of the <MUMPS length specification> and whose character set is implementation-defined.

6) INT describes an exact numeric variable. The equivalent SQL data type is INTEGER.

7) DEC describes an exact numeric variable. The <scale> shall not be greater than the <precision>. The equivalent SQL data type is DECIMAL with the same <precision> and <scale>.

8) REAL describes an approximate numeric variable. The equivalent SQL data type is REAL.

9) If an <embedded SQL MUMPS program> contains neither a <MUMPS variable definition> that specifies SQLSTATE and that is defined with a <MUMPS character variable > that specifies VARCHAR with a <MUMPS length specification> that specifies 5, nor a <MUMPS variable definition> that specifies SQLCODE and that is defined with a <MUMPS numeric variable> that specifies INT, then it is assumed that the <embedded SQL MUMPS program> contains a variable named SQLCODE defined with a data type of INT.

   Note: SQLSTATE is the preferred status parameter. The SQLCODE status parameter is a deprecated feature that is supported for compatibility with earlier versions of this
   ANSI American
   ISO International
   Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 21.1, "<embedded SQL host program>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall contain no <embedded SQL MUMPS program>.

21.8 <embedded SQL Pascal program>

Function

Specify an <embedded SQL Pascal program>.

Format

```
<embedded SQL Pascal program> ::= !! See the Syntax Rules.

<Pascal variable definition> ::=
       <Pascal host identifier> [ { <comma> <Pascal host identifier> }... ] <colon>
         <Pascal type specification> <semicolon>

<Pascal host identifier> ::= !! See the Syntax Rules.

<Pascal type specification> ::=
         PACKED ARRAY <left bracket> 1 <double period> <length> <right bracket>
           OF CHAR
             [ CHARACTER SET [ IS ] <character set specification> ]
       | PACKED ARRAY <left bracket> 1 <double period> <length> <right bracket>
           OF BIT
       | INTEGER
       | REAL
       | CHAR [ CHARACTER SET [ IS ] <character set specification> ]
       | BIT
       | BOOLEAN
```

Syntax Rules

1) An <embedded SQL Pascal program> is a compilation unit that consists of Pascal text and SQL text. The Pascal text shall conform to to one of the the Pascal standards
   ANSI ANSI/IEEE 770/X3.97 and ANSI/IEEE 770/X3.160.
   ISO ISO 7185 and ISO/IEC 10206.
   The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever a Pascal statement may be specified. An <embedded SQL statement> may be prefixed by a Pascal label.

3) A <Pascal host identifier> is a Pascal variable-identifier whose applied instance denotes a defining instance within an <embedded SQL begin declare> and an <embedded SQL end declare>.

4) A <Pascal variable definition> defines one or more <Pascal host identifier>s.

5) A <Pascal variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL Pascal program> (see the Syntax Rules of Subclause 21.1, "<embedded SQL host program>").

   a) Any optional CHARACTER SET specification shall be removed from the PACKED ARRAY OF CHAR or CHAR alternatives of a <Pascal type specification>.

b) The <length> specified in the PACKED ARRAY OF CHAR alternative of any <Pascal type specification> that contained a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <Pascal host identifier> specified in the containing <Pascal variable definition>.

c) If any <Pascal type sepcification> specifies the syntax ''CHAR'' and contains a CHARACTER SET specification, then let *L* be a length equal to the length in octets of *PN* and *PN* be the <Pascal host identifier> specified in the containing <Pascal variable definition>. If *L* is greater than 1, then ''CHAR'' shall be replaced by ''PACKED ARRAY [1..*L*] OF CHAR''.

d) The syntax ''BIT'' shall be replaced by ''CHAR'' in any PACKED ARRAY OF BIT or BIT alternatives of a <Pascal type specification>.

e) The <length> specified in any PACKED ARRAY OF BIT alternative in a <Pascal type specification> shall be replaced by a length equal to the smallest integer not less than *L*/*B*, as defined in the Syntax Rules of this Subclause.

The modified <Pascal variable definition> shall be a valid Pascal variable-declaration in the program derived from the <embedded SQL Pascal program>.

6) CHAR specified without a CHARACTER SET specification is the ordinal-type-identifier of PASCAL. The equivalent SQL data type is CHARACTER with length 1.

7) BIT describes a single-bit variable. It is mapped to a Pascal CHAR ordinal-type-identifier whose most significant bit contains either 0 or 1 and whose least significant bits contain 0. The equivalent SQL data type is BIT with length 1.

8) PACKED ARRAY [1..<length>] OF CHAR describes a character string having 2 or more components of the simple type CHAR. The equivalent SQL data type is CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

9) PACKED ARRAY [1..<length>] OF BIT describes a bit string variable. Let *B* be the number of bits in a Pascal CHAR and let *L* be the <length> of the bit string variable. The length of an equivalent Pascal character variable is the smallest integer not less than *L*/*B*. The equivalent SQL data type is BIT whose length is the <length> of the bit string variable.

10) INTEGER describes an exact numeric variable. The equivalent SQL data type is INTEGER.

11) REAL describes an approximate numeric variable. The equivalent SQL data type is REAL.

12) BOOLEAN describes a boolean variable. The equivalent SQL data type is BOOLEAN.

13) If an <embedded SQL Pascal program> contains neither a <Pascal variable definition> that specifies SQLSTATE and that is defined with a <Pascal type specification> that specifies PACKED ARRAY [1..<length>] OF CHAR with <length> 5, nor a <Pascal variable definition> that specifies SQLCODE and that is defined with a <Pascal type specification> that specifies INTEGER, then it is assumed that the <embedded SQL Pascal program> contains a variable named SQLCODE defined with a data type of INTEGER.

Note: SQLSTATE is the preferred status parameter. The SQLCODE status parameter is a deprecated feature that is supported for compatibility with earlier versions of this
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 21.1, "<embedded SQL host program>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <Pascal type specification> shall not specify BIT or PACKED ARRAY [1..<length>] OF BIT.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <Pascal type specification> shall not contain a <character set specification>.

21.9 <embedded SQL PL/I program>

Function

Specify an <embedded SQL PL/I program>.

Format

```
<embedded SQL PL/I program> ::= !! See the Syntax Rules.

<PL/I variable definition> ::=
       {DCL | DECLARE}
          {   <PL/I host identifier>
            | <left paren> <PL/I host identifier>
                  [ { <comma> <PL/I host identifier> }... ] <right paren> }
       <PL/I type specification>
       [ <character representation>... ] <semicolon>

<PL/I host identifier> ::= !! See the Syntax Rules.

<PL/I type specification> ::=
         { CHAR | CHARACTER } [ VARYING ] <left paren><length><right paren>
             [ CHARACTER SET [ IS ] <character set specification> ]
       | BIT [ VARYING ] <left paren><length><right paren>
       | <PL/I type fixed decimal> <left paren> <precision>
             [ <comma> <scale> ] <right paren>
       | <PL/I type fixed binary> [ <left paren> <precision> <right paren> ]
       | <PL/I type float binary> <left paren> <precision> <right paren>

<PL/I type fixed decimal> ::=
         { DEC | DECIMAL } FIXED
       | FIXED { DEC | DECIMAL }

<PL/I type fixed binary> ::=
         { BIN | BINARY } FIXED
       | FIXED { BIN | BINARY }

<PL/I type float binary> ::=
         { BIN | BINARY } FLOAT
       | FLOAT { BIN | BINARY }
```

Syntax Rules

1) An <embedded SQL PL/I program> is a compilation unit that consists of PL/I text and SQL text. The PL/I text shall conform to to the PL/I standard
   ANSI ANSI X3.53.
   ISO ISO 6160.
   The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever a PL/I statement may be specified within a procedure block. If the PL/I statement could include a label prefix, the <embedded SQL statement> may be immediately preceded by a label prefix.

3) A <PL/I host identifier> is any valid PL/I variable identifier. A <PL/I host identifier> shall be contained in an <embedded SQL PL/I program>.

4) A <PL/I variable definition> defines one or more host variables.

5) A <PL/I variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL PL/I program> (see the Syntax Rules of Subclause 21.1, "<embedded SQL host program>").

   a) Any optional CHARACTER SET specification shall be removed from the CHARACTER or CHARACTER VARYING alternatives of a <PL/I type specification>.

   b) The <length> specified in the CHARACTER or CHARACTER VARYING alternatives of any <PL/I type specification> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <PL/I host identifier> specified in the containing <PL/I variable definition>.

   The modified <PL/I variable definition> shall be a valid PL/I data declaration in the program derived from the <embedded SQL PL/I program>.

6) A <PL/I variable definition> shall specify a scalar variable, not an array or structure.

7) The optional <character representation> sequence in a <PL/I variable definition> may specify an INITIAL clause. Whether other clauses may be specified is implementation-defined. The <character representation> sequence shall be such that the <PL/I variable definition> is a valid PL/I DECLARE statement.

8) CHARACTER describes a character string variable whose equivalent SQL data type has the character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

   Case:

   a) If VARYING is not specified, then the length of the variable is fixed. The equivalent SQL data type is CHARACTER with the same length.

   b) If VARYING is specified, then the variable is of variable length, with maximum size the value of <length>. The equivalent SQL data type is CHARACTER VARYING with the same maximum length.

9) BIT describes a bit string variable.

   Case:

   a) If VARYING is not specified, then the length of the variable is fixed. The equivalent SQL data type is BIT with the same length. length. If the length of the variable is 1, then the equivalent SQL data type may be BOOLEAN.

   b) If VARYING is specified, then the variable is of variable length with maximum size of the value of <length>. The equivalent SQL data type is BIT VARYING with the same maximum length.

10) FIXED DECIMAL describes an exact numeric variable. The <scale> shall not be greater than the <precision>. The equivalent SQL data type is DECIMAL with the same <precision> and <scale>.

11) FIXED BINARY describes an exact numeric variable. The equivalent SQL data type is SMALLINT or INTEGER.

12) FLOAT BINARY describes an approximate numeric variable. The equivalent SQL data type is FLOAT with the same <precision>.

13) If an <embedded SQL PL/I program> contains neither a <PL/I variable definition> that specifies SQLSTATE and that is defined with a <PL/I type specification> that specifies CHAR or CHARACTER with <length> 5 and does not specify VARYING, nor a <PL/I variable definition> that specifies SQLCODE and that is defined with a <PL/I type specification> that specifies FIXED BIN(*PP*) or FIXED BINARY(*PP*), where *PP* is the implementation-defined precision specified for a PL/I SQLCODE parameter in Subclause 10.3, "<routine invocation>", then it is assumed that the <embedded SQL PL/I program> contains a variable named SQLCODE defined with a data type of FIXED BINARY(*PP*), where *PP* is the implementation-defined precision specified for a PL/I SQLCODE parameter in Subclause 10.3, "<routine invocation>".

Note: SQLSTATE is the preferred status parameter. The SQLCODE status parameter is a deprecated feature that is supported for compatibility with earlier versions of this
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 21.1, "<embedded SQL host program>"

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <PL/I type specification> shall not specify BIT or BIT VARYING.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <PL/I type specification> shall not specify CHARACTER VARYING.

   b) A <PL/I type specification> shall not contain a <character set specification>. <character set specification>.

22 Direct invocation of SQL

22.1 <direct SQL statement>

Function

Specify direct execution of SQL.

Format

```
<direct SQL statement> ::=
       <directly executable statement> <semicolon>

<directly executable statement> ::=
         <direct SQL data statement>
       | <SQL schema statement>
       | <SQL transaction statement>
       | <SQL connection statement>
       | <SQL session statement>
       | <direct implementation-defined statement>

<direct SQL data statement> ::=
         <delete statement: searched>
       | <direct select statement: multiple rows>
       | <insert statement>
       | <update statement: searched>
       | <temporary table declaration>

<direct implementation-defined statement> ::= !! See the Syntax Rules
```

Syntax Rules

1) The <direct SQL data statement> shall not contain any

ANSI Only–SQL3

parameter reference, SQL variable reference, <dynamic parameter specification>, or <embedded variable specification>.

ISO Only—caused by ANSI changes not yet considered by ISO

<target specification>.

2) The <value specification> that represents the null value is implementation-defined.

3) The Format and Syntax Rules for <direct implementation-defined statement> are implementation-defined.

Access Rules

1) The Access Rules for <direct implementation-defined statement> are implementation-defined.

General Rules

1) The following <direct SQL statement>s are transaction-initiating <direct SQL statement>s:

   a) <direct SQL statement>s that are transaction-initiating <SQL procedure statement>s;

   b) <direct select statement: multiple rows>; and

   c) <direct implementation-defined statement>s that are transaction-initiating.

2) After the last invocation of an SQL-statement by an SQL-agent in an SQL-session:

   a) A <rollback statement> or a <commit statement> is effectively executed. If an unrecoverable error has occurred, or if the direct invocation of SQL terminated unexpectedly, or if any constraint is not satisfied, then a <rollback statement> is performed. Otherwise, the choice of which of these SQL-statements to perform is implementation-dependent. The determination of whether a direct invocation of SQL has terminated unexpectedly is implementation-dependent.

   b) Let *D* be the <descriptor name> of any system descriptor area that is currently allocated within the current SQL-session. A <deallocate descriptor statement> that specifies

      DEALLOCATE DESCRIPTOR *D*

      is effectively executed.

   c) All SQL-sessions associated with the SQL-agent are terminated.

3) Let *S* be the <direct SQL statement>.

4) The current <authorization identifier> for privilege determination for the execution of *S* is the SQL-session <authorization identifier>.

5) If *S* does not conform to the Format, Syntax Rules, and Access Rules for a <direct SQL statement>, then an exception condition is raised: *syntax error or access rule violation in direct SQL statement*.

6) When *S* is invoked by the SQL-agent:

   Case:

   a) If *S* is an <SQL connection statement>, then:

      i) The diagnostics area is emptied.

      ii) *S* is executed.

      iii) If *S* successfully initiated or resumed an SQL-session, then subsequent invocations of a <direct SQL statement> by the SQL-agent are associated with that SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

   b) Otherwise:

      i) If no SQL-session is current for the SQL-agent, then

Case:

1) If the SQL-agent has not executed an <SQL connection statement> and there is no default SQL-session associated with the SQL-agent, then the following <connect statement> is effectively executed:

   CONNECT TO DEFAULT

2) If the SQL-agent has not executed an <SQL connection statement> and there is a default SQL-session associated with the SQL-agent, then the following <set connection statement> is effectively executed:

   SET CONNECTION DEFAULT

3) Otherwise, an exception condition is raised: *connection exception—connection does not exist*.

Subsequent calls to a <routine> or invocations of a <direct SQL statement> by the SQL-agent are associated with the SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

ii) If an SQL-transaction is active for the SQL-agent, then *S* is associated with that SQL-transaction. If *S* is a <direct implementation-defined statement>, then it is implementation-defined whether or not *S* may be associated with an active SQL-transaction; if not, then an exception condition is raised: *invalid transaction state*.

iii) If no SQL-transaction is active for the SQL-agent, then

1) Case:

   A) If *S* is a transaction-initiating <direct SQL statement>, then an SQL-transaction is initiated.

   B) If *S* is a <direct implementation-defined statement>, then it is implementation-defined whether or not *S* initiates an SQL-transaction. If an implementation defines *S* to be transaction-initiating, then an SQL-transaction is initiated.

2) If *S* initiated an SQL-transaction, then:

   A) Let *T* be the SQL-transaction initiated by *S*.

   B) *T* is associated with this invocation and any subsequent invocations of <direct SQL statement>s or calls to a <routine> by the SQL-agent until the SQL-agent terminates *T*.

   C) Case:

      I) If a <set transaction statement> has been executed since the termination of the last SQL-transaction in the SQL-session (or if there has been no previous SQL-transaction in the SQL-session and a <set transaction statement> has been executed), then the access mode, constraint mode, and isolation level of *T* are set as specified by the <set transaction statement>.

      II) Otherwise, the access mode, constraint mode for all constraints, and isolation level for *T* are *read-write*, *immediate*, and SERIALIZABLE, respectively.

   D) *T* is associated with the SQL-session.

iv) If *S* contains an <SQL schema statement> and the access mode of the current SQL-transaction is *read-only*, then an exception condition is raised: *invalid transaction state*.

v) The diagnostics area is emptied.

vi) *S* is executed.

7) If the execution of a <direct SQL data statement> occurs within the same SQL-transaction as the execution of an SQL-schema statement and this is not allowed by the SQL-implementation, then an exception condition is raised: *invalid transaction state*.

8) Case:

a) If *S* executed successfully, then either a completion condition is raised: *successful execution*, or a completion condition is raised: *warning*, or a completion condition is raised: *no data*, as determined by the General Rules in this and other Subclauses of this
ANSI American
ISO International
Standard.

b) If *S* did not execute successfully, then all changes made to SQL-data or schemas by the execution of *S* are canceled and an exception condition is raised as determined by the General Rules in this and other Subclauses of this
ANSI American
ISO International
Standard.

Note: The method of raising a condition is implementation-defined.

9) Diagnostics information resulting from the execution of *S* is placed into the diagnostics area as specified in Clause 20, "Diagnostics management".

Note: The method of accessing the diagnostics information is implementation-defined, but does not alter the contents of the diagnostics area.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <direct SQL statement> shall not be an <SQL schema statement>.

22.2 <direct select statement: multiple rows>

Function

Specify a statement to retrieve multiple rows from a specified table.

Format

```
<direct select statement: multiple rows> ::=
       <query expression> [ <order by clause> ]
```

Syntax Rules

1) All Syntax Rules of Subclause 7.13, "<query expression>", apply to the <direct select statement: multiple rows>.

2) The <query expression> or <order by clause> of a <direct select statement: multiple rows> shall not contain any <value specification> other than a <literal>,
ISO USER,
CURRENT_USER, SESSION_USER, or SYSTEM_USER.

3) Let *T* be the table specified by the <query expression>.

4) If ORDER BY is specified, then each <sort specification> in the <order by clause> shall identify a column of *T*.

   Case:

   a) If a <sort specification> contains a <column name>, then *T* shall contain exactly one column with that <column name> and the <sort specification> identifies that column.

   b) If a <sort specification> contains an <unsigned integer>, then the <unsigned integer> shall be greater than 0 and not greater than the degree of *T*. The <sort specification> identifies the column of *T* with the ordinal position specified by the <unsigned integer>.

Access Rules

None.

General Rules

1) All General Rules of Subclause 7.13, "<query expression>", apply to the <direct select statement: multiple rows>.

2) Let *Q* be the result of the <query expression>.

3) If *Q* is empty, then a completion condition is raised: *no data*.

4) If an <order by clause> is not specified, then the ordering of the rows of *Q* is implementation-dependent.

5) If an <order by clause> is specified, then the ordering of rows of the result is effectively determined by the <order by clause> as follows:

   a) Each <sort specification> specifies the sort direction for the corresponding sort key K_i. If ASC is specified or implied in the *i*-th <sort specification>, then the sort direction for K_i is ascending and the applicable <comp op> is the <less than operator>. Otherwise, the sort direction for K_i is descending and the applicable <comp op> is the <greater than operator>.

   b) Let *X* and *Y* be distinct rows in the result table, and let XV_i and YV_i be the values of K_i in these rows, respectively. The relative position of rows *X* and *Y* in the result is determined by comparing XV_i and YV_i according to the rules of Subclause 8.2, "<comparison predicate>", where the <comp op> is the applicable <comp op> for K_i, with the following special treatment of null values. Whether a sort key value that is null is considered greater or less than a non-null value is implementation-defined, but all sort key values that are null shall either be considered greater than all non-null values or be considered less than all non-null values. XV_i is said to *precede* YV_i if the value of the <comparison predicate> ''XV_i <comp op> YV_i'' is true for the applicable <comp op>.

   c) In the result table, the relative position of row *X* is before row *Y* if and only if XV_n precedes YV_n for some *n* greater than 0 and less than the number of <sort specification>s and $XV_i = YV_i$ for all $i < n$. The relative order of two rows for which $XV_i = YV_i$ for all *i* is implementation-dependent.

6) If *Q* is not empty, then *Q* is returned. The method of returning *Q* is implementation-defined.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

23 Information Schema and Definition Schema

23.1 Introduction

The views of the Information Schema are viewed tables defined in terms of the base tables and views of the Definition Schema. The only purpose of the Definition Schema is to provide a data model to support the Information Schema and to assist understanding. An implementation need do no more than simulate the existence of the Definition Schema, as viewed through the Information Schema views.

The Information Schema views are defined as being in a schema named INFORMATION_SCHEMA, enabling these views to be accessed in the same way as any other tables in any other schema. SELECT on all of these views is granted to PUBLIC WITH GRANT OPTION, so that they can be queried by any user and so that SELECT privilege can be further granted on views that reference these Information Schema views. No other privilege is granted on them, so they cannot be updated.

The Information Schema also contains a small number of domains on which the columns of the Definition Schema are based. USAGE on all these domains is granted to PUBLIC WITH GRANT OPTION, so that they can be used by any user.

An implementation may define objects that are associated with INFORMATION_SCHEMA that are not defined in this Clause. An implementation may also add columns to tables that are defined in this Clause.

The Definition Schema base tables and views are defined as being in a schema named DEFINITION_SCHEMA. Because <unqualified schema name>s are prohibited from specifying DEFINITION_SCHEMA, the Definition Schema cannot be accessed in an SQL-statement.

Note: The Information Schema tables may be supposed to be represented in the Definition Schema in the same way as any other tables, and are hence self-describing.

Note: The Information Schema is a definition of the SQL data model, specified as an SQL-schema, in terms of <SQL schema statement>s as defined in this
ANSI American
ISO International
standard. Constraints defined in this Clause are not actual SQL constraints.

<identifier>s are represented in the tables of the Information Schema by <string value expression>s corresponding to their <identifier body>s (in the case of <regular identifier>s) or their <delimited identifier body>s (in the case of <delimited identifier>s). Comparison of <identifier>s is defined in Subclause 8.2, "<comparison predicate>". Where an <identifier> has many equivalent forms, the one encountered at definition time is stored (of course, any lower case letters appearing in a <regular identifier> will have been converted to the corresponding upper case letter before the <identifier> is stored in any table of the Information Schema).

****Editor's Note****

The redundancy in the Description of some of the Information Schema tables has been identified as a Possible Problem. See Possible Problem **091** in the Editor's Notes.

23.2 Information Schema

23.2.1 INFORMATION_SCHEMA Schema

Function

Identify the schema that is to contain the Information Schema tables.

Definition

```
CREATE SCHEMA INFORMATION_SCHEMA
       AUTHORIZATION INFORMATION_SCHEMA
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.2 INFORMATION_SCHEMA_CATALOG_NAME base table

Function

Identify the catalog that contains the Information Schema.

Definition

```
CREATE TABLE INFORMATION_SCHEMA_CATALOG_NAME
       ( CATALOG_NAME        SQL_IDENTIFIER,
         CONSTRAINT INFORMATION_SCHEMA_CATALOG_NAME_PRIMARY_KEY
           PRIMARY KEY ( CATALOG_NAME )
         CONSTRAINT INFORMATION_SCHEMA_CATALOG_NAME_CHECK
           CHECK ( ( SELECT COUNT(*) FROM INFORMATION_SCHEMA_CATALOG_NAME ) = 1 ) <
        )
```

Description

1) The value of CATALOG_NAME is the name of the catalog in which this Information Schema resides.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.3 INFORMATION_SCHEMA_CATALOG_NAME_CARDINALITY assertion

Function

Ensure that there is exactly one row in the INFORMATION_SCHEMA_CATALOG_NAME table.

Definition

```
CREATE ASSERTION INFORMATION_SCHEMA_CATALOG_NAME_CARDINALITY
       CHECK ( 1 = ( SELECT COUNT(*)
                       FROM INFORMATION_SCHEMA_CATALOG_NAME ) )
```

Description

1) The INFORMATION_SCHEMA_CATALOG_NAME_CARDINALITY assertion ensures that there is exactly one row in the INFORMATION_SCHEMA_CATALOG_NAME table.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.4 SCHEMATA view

Function

Identify the schemata that are owned by a given user.

Definition

```
CREATE VIEW SCHEMATA
       AS SELECT
           CATALOG_NAME, SCHEMA_NAME, SCHEMA_OWNER,
           DEFAULT_CHARACTER_SET_CATALOG, DEFAULT_CHARACTER_SET_SCHEMA,
           DEFAULT_CHARACTER_SET_NAME
         FROM DEFINITION_SCHEMA.SCHEMATA
         WHERE SCHEMA_OWNER = CURRENT_USER
           AND CATALOG_NAME
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.5 DOMAINS view

Function

Identify the domains defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW DOMAINS
       AS SELECT DISTINCT
         DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,
         DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
         COLLATION_CATALOG, COLLATION_SCHEMA,COLLATION_NAME,
         CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
         NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE,
```

ANSI Only--SQL3

```
         DATETIME_PRECISION,INTERVAL_CODE, INTERVAL_PRECISION, DOMAIN_DEFAULT
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
         DATETIME_PRECISION, DOMAIN_DEFAULT
```

```
         ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA, ABSTRACT_DATA_TYPE_NAME,

         NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME
       FROM DEFINITION_SCHEMA.DOMAINS
            JOIN
              DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D
              LEFT JOIN
              DEFINITION_SCHEMA.COLLATIONS AS S
              USING ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
            ON
              ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME, '' )
              = ( TABLE_OR_DOMAIN_CATALOG, TABLE_OR_DOMAIN_SCHEMA,
                    TABLE_OR_DOMAIN_NAME, COLUMN_NAME ) )
         WHERE
               ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME, 'DOMAIN' )
                 IN
                 ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                   FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
                     WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
               OR
               ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME ) IN

                     FROM COLUMNS ) )
           AND DOMAIN_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.6 DOMAIN_CONSTRAINTS view

Function

Identify the domain constraints of domains in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW DOMAIN_CONSTRAINTS
       AS SELECT DISTINCT
         CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
         DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,
         IS_DEFERRABLE, INITIALLY_DEFERRED
       FROM DEFINITION_SCHEMA.DOMAIN_CONSTRAINTS
            JOIN
              DEFINITION_SCHEMA.SCHEMATA AS S
                 ON
                 ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
                 = ( S.CATALOG_NAME, SCHEMA_NAME S) )
         WHERE
           SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.7 TABLES view

Function

Identify the tables defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW TABLES
       AS SELECT
         TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, TABLE_TYPE
       FROM DEFINITION_SCHEMA.TABLES
           WHERE ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
             IN (
                 SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                  FROM DEFINITION_SCHEMA.TABLE_PRIVILEGES
                    WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                 UNION
                 SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                  FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES
                    WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
           AND TABLE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.8 VIEWS view

Function

Identify the viewed tables defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW VIEWS
       AS SELECT
         TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
         CASE WHEN ( TABLE_CATALOG, TABLE_SCHEMA, CURRENT_USER )
                     IN ( SELECT CATALOG_NAME, SCHEMA_NAME, SCHEMA_OWNER
                                 FROM DEFINITION_SCHEMA.SCHEMATA )
              THEN VIEW_DEFINITION
              ELSE NULL
         END AS VIEW_DEFINITION,
           CHECK_OPTION, IS_UPDATABLE
       FROM DEFINITION_SCHEMA.VIEWS
           WHERE ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
             IN ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                  FROM TABLES )
           AND TABLE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.9 COLUMNS view

Function

Identify the columns of tables defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW COLUMNS
       AS SELECT DISTINCT
         TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
         C.COLUMN_NAME, ORDINAL_POSITION,
         CASEWHEN EXISTS ( SELECT *
                FROM DEFINITION_SCHEMA.SCHEMATA AS S
                WHERE ( TABLE_CATALOG, TABLE_SCHEMA )
                    = (S.CATALOG_NAME, S.SCHEMA_NAME )
                  AND SCHEMA_OWNER = USER )
               THEN COLUMN_DEFAULT
              ELSE NULL
         END AS COLUMN_DEFAULT,
         IS_NULLABLE,
         COALESCE (D1.DATA_TYPE, D2.DATA_TYPE) AS DATA_TYPE,
         COALESCE (D1.CHARACTER_MAXIMUM_LENGTH, D2.CHARACTER_MAXIMUM_LENGTH)
           AS CHARACTER_MAXIMUM_LENGTH,
         COALESCE (D1.CHARACTER_OCTET_LENGTH, D2.CHARACTER_OCTET_LENGTH)
                  AS CHARACTER_OCTET_LENGTH,
         COALESCE (D1.NUMERIC_PRECISION, D2.NUMERIC_PRECISION)
                  AS NUMERIC_PRECISION,
         COALESCE (D1.NUMERIC_PRECISION_RADIX, D2.NUMERIC_PRECISION_RADIX)
                  AS NUMERIC_PRECISION_RADIX,
         COALESCE (D1.NUMERIC_SCALE, D2.NUMERIC_SCALE) AS NUMERIC_SCALE,
         COALESCE (D1.DATETIME_PRECISION, D2.DATETIME_PRECISION) AS DATETIME_PRECISION,
```

ANSI Only--SQL3

```
         COALESCE (D1.INTERVAL_CODE, D3.INTERVAL_CODE) AS INTERVAL_CODE
         COALESCE (D1.INTERVAL_PRECISION, D3.INTERVAL_PRECISION) AS INTERVAL_PRECISION
```

```
         COALESCE (C1.CHARACTER_SET_CATALOG, C2.CHARACTER_SET_CATALOG)
                  AS CHARACTER_SET_CATALOG,
         COALESCE (C1.CHARACTER_SET_SCHEMA, C2.CHARACTER_SET_SCHEMA)
                  AS CHARACTER_SET_SCHEMA,
         COALESCE (C1.CHARACTER_SET_NAME, C2.CHARACTER_SET_NAME) AS CHARACTER_SET_NAME,
         COALESCE (D1.COLLATION_CATALOG, D2.COLLATION_CATALOG) AS COLLATION_CATALOG,
         COALESCE (D1.COLLATION_SCHEMA, D2.COLLATION_SCHEMA) AS COLLATION_SCHEMA,
         COALESCE (D1.COLLATION_NAME, D2.COLLATION_NAME) AS COLLATION_NAME,
         DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,
         COALESCE (D1.ABSTRACT_DATA_TYPE_CATALOG, D2.ABSTRACT_DATA_TYPE_CATALOG)
                  AS ABSTRACT_DATA_TYPE_CATALOG,
         COALESCE (D1.ABSTRACT_DATA_TYPE_SCHEMA, D2.ABSTRACT_DATA_TYPE_SCHEMA)
                  AS ABSTRACT_DATA_TYPE_SCHEMA,
         COALESCE (D1.ABSTRACT_DATA_TYPE_NAME, D2.ABSTRACT_DATA_TYPE_NAME)
                  AS ABSTRACT_DATA_TYPE_NAME
         COALESCE (D1.NULL_CLASS_CATALOG, D2.NULL_CLASS_CATALOG) AS NULL_CLASS_CATALOG,
         COALESCE (D1.NULL_CLASS_SCHEMA, D2.NULL_CLASS_SCHEMA) AS NULL_CLASS_SCHEMA,
         COALESCE (D1.NULL_CLASS_NAME, D2.NULL_CLASS_NAME) AS NULL_CLASS_NAME
       FROM DEFINITION_SCHEMA.COLUMNS AS C
           LEFT JOIN
             DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D1
               LEFT JOIN
```

```
      DEFINITION_SCHEMA.COLLATIONS AS C1
      ON
      ( ( C1.COLLATION_CATALOG, C1.COLLATION_SCHEMA, C1.COLLATION_NAME )
      = ( D1.COLLATION_CATALOG, D1.COLLATION_SCHEMA, D1.COLLATION_NAME ) )
ON
  ( ( C.TABLE_CATALOG, C.TABLE_SCHEMA, C.TABLE_NAME, C.COLUMN_NAME )
  = ( D1.TABLE_OR_DOMAIN_CATALOG, D1.TABLE_OR_DOMAIN_SCHEMA,
      D1.TABLE_OR_DOMAIN_NAME, D1.COLUMN_NAME ) )
LEFT JOIN
  DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D2
    LEFT JOIN
      DEFINITION_SCHEMA.COLLATIONS AS C2
      ON
      ( ( C2.COLLATION_CATALOG, C2.COLLATION_SCHEMA, C2.COLLATION_NAME )
      = ( D2.COLLATION_CATALOG, D2.COLLATION_SCHEMA, D2.COLLATION_NAME ) )
ON
  ( ( C.DOMAIN_CATALOG, C.DOMAIN_SCHEMA, C.DOMAIN_NAME )
  = ( D2.TABLE_OR_DOMAIN_CATALOG, D2.TABLE_OR_DOMAIN_SCHEMA,
      D2.TABLE_OR_DOMAIN_NAME ) )
WHERE ( C.TABLE_CATALOG, C.TABLE_SCHEMA, C.TABLE_NAME, C.COLUMN_NAME )
  IN
    ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
        FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES
          WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
AND C.TABLE_CATALOG
  = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.10 ABSTRACT_DATA_TYPES view

Function

Identify the abstract data types that are accessible to a given user.

Definition

```
CREATE VIEW ABSTRACT_DATA_TYPES
       AS SELECT * FROM
          (
            SELECT ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                   ABSTRACT_DATA_TYPE_NAME,
                   ABSTRACT_DATA_TYPE_EQUALS, ABSTRACT_DATA_TYPE_LESS_THAN
              FROM DEFINITION_SCHEMA.ABSTRACT_DATA_TYPES
                WHERE ABSTRACT_DATA_TYPE_SCHEMA = USER
                      OR
                      ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                        ABSTRACT_DATA_TYPE_NAME ) IN
                        ( SELECT ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                                 ABSTRACT_DATA_TYPE_NAME
                            FROM DEFINITION_SCHEMA.ABSTRACT_DATA_TYPE_PRIVILEGES
                              WHERE GRANTEE IN ( 'PUBLIC', USER ) )
            JOIN
            SELECT ABSTRACT_DATA_TYPE_OPERATION_NAME, ABSTRACT_DATA_TYPE_PARAMETERS,
                   ABSTRACT_DATA_TYPE_RETURNS, ABSTRACT_DATA_TYPE_ENTRY,
                   ABSTRACT_DATA_TYPE_ERROR
              FROM DEFINITION_SCHEMA.ABSTRACT_DATA_TYPE_OPERATIONS
                ON FOREIGN KEY
          )
          GROUP BY ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                   ABSTRACT_DATA_TYPE_NAME, ABSTRACT_DATA_TYPE_LANGUAGE
```

****Editor's Note****

This view refers to the ABSTRACT_DATA_TYPE_OPERATIONS base table (see the FROM clause), but that base table doesn't seem to exist!

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

23.2.11 NULL_CLASSES view

Function

Identify the null classes that are accessible to a given user.

Format

```
CREATE VIEW NULL_CLASSES
       AS SELECT DISTINCT
       NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME
       FROM DEFINITION_SCHEMA.NULL_CLASSES
         WHERE NULL_CLASS_SCHEMA = USER
               OR
               ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA,
                 NULL_CLASS_NAME, 'NULL_CLASS' ) IN
               ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                   FROM _DEFINITION_SCHEMA.USAGE_PRIVILEGES
                   WHERE GRANTEE = 'PUBLIC'
                         OR
                         GRANTEE = USER )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.12 NULL_STATES view

Function

Identify the null states that are accessible to a given user.

Format

```
CREATE VIEW NULL_STATES
       AS SELECT DISTINCT
         NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME
         NULL_STATE_NAME, ORDINAL_POSITION
       FROM DEFINITION_SCHEMA.NULL_STATES
         WHERE NULL_CLASS_SCHEMA = USER
               OR
               ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA,
                 NULL_CLASS_NAME, 'NULL_CLASS' ) IN
               ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                   FROM _DEFINITION_SCHEMA.USAGE_PRIVILEGES
                   WHERE GRANTEE = 'PUBLIC'
                         OR
                         GRANTEE = USER )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.13 TABLE_PRIVILEGES view

Function

Identify the privileges on tables defined in this catalog that are available to or granted by a given user.

Definition

```
CREATE VIEW TABLE_PRIVILEGES
       AS SELECT
         GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
           PRIVILEGE_TYPE, IS_GRANTABLE
       FROM DEFINITION_SCHEMA.TABLE_PRIVILEGES
           WHERE   GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                OR GRANTOR = CURRENT_USER
           AND TABLE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.14 COLUMN_PRIVILEGES view

Function

Identify the privileges on columns of tables defined in this catalog that are available to or granted by a given user.

Definition

```
CREATE VIEW COLUMN_PRIVILEGES
       AS SELECT
         GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME,
           PRIVILEGE_TYPE, IS_GRANTABLE
       FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES
           WHERE   GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                OR GRANTOR = CURRENT_USER
           AND TABLE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.15 USAGE_PRIVILEGES view

Function

Identify the USAGE privileges on objects defined in this catalog that are available to or granted by a given user.

Definition

```
CREATE VIEW USAGE_PRIVILEGES
       AS SELECT
         GRANTOR, GRANTEE, OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
           OBJECT_TYPE, 'USAGE' AS PRIVILEGE_TYPE, IS_GRANTABLE
       FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
           WHERE   GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                OR GRANTOR = CURRENT_USER
           AND OBJECT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.16 ABSTRACT_DATA_TYPE_PRIVILEGES view

Function

Identify the abstract data types defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW ABSTRACT_DATA_TYPE_PRIVILEGES
       AS SELECT
             GRANTEE, ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
             ABSTRACT_DATA_TYPE_NAME, GRANTOR, PRIVILEGE_TYPE, IS_GRANTABLE
         FROM DEFINITION_SCHEMA.ABSTRACT_DATA_TYPE_PRIVILEGES
           WHERE GRANTEE IN ( 'PUBLIC', USER )
              OR GRANTOR = USER
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

23.2.17 ABSTRACT_DATA_TYPE_IMPACT_COLUMN view

Function

Identify the columns that are dependent on an abstract data type.

Definition

```
CREATE VIEW ABSTRACT_DATA_TYPE_IMPACT_COLUMN
       AS SELECT
           ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
           ABSTRACT_DATA_TYPE_NAME,
           TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
         FROM DEFINITION_SCHEMA.COLUMNS
           WHERE ABSTRACT_DATA_TYPE_NAME IS NOT NULL
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

23.2.18 ABSTRACT_DATA_TYPE_IMPACT_DOMAIN view

Function

Identify the domains that are dependent on an abstract data type.

Definition

```
CREATE VIEW ABSTRACT_DATA_TYPE_IMPACT_DOMAIN
       AS SELECT
             ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
             ABSTRACT_DATA_TYPE_NAME,
             DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME
         FROM DEFINITION_SCHEMA.DOMAINS
           WHERE ABSTRACT_DATA_TYPE_NAME IS NOT NULL
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

23.2.19 TABLE_CONSTRAINTS view

Function

Identify the table constraints defined in this catalog that are owned by a given user.

Definition

```
CREATE VIEW TABLE_CONSTRAINTS
       AS SELECT
           CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
           TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
           CONSTRAINT_TYPE, IS_DEFERRABLE, INITIALLY_DEFERRED
         FROM DEFINITION_SCHEMA.TABLE_CONSTRAINTS
              JOIN
              DEFINITION_SCHEMA.SCHEMATA S
              ON
               ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
               = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
            WHERE SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.20 REFERENTIAL_CONSTRAINTS view

Function

Identify the referential constraints defined in this catalog that are owned by a given user.

Definition

```
CREATE VIEW REFERENTIAL_CONSTRAINTS
       AS SELECT
           CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
           UNIQUE_CONSTRAINT_CATALOG, UNIQUE_CONSTRAINT_SCHEMA, UNIQUE_CONSTRAINT_NAME,

           MATCH_OPTION, UPDATE_RULE, DELETE_RULE
         FROM DEFINITION_SCHEMA.REFERENTIAL_CONSTRAINTS
              JOIN
              DEFINITION_SCHEMA.SCHEMATA S
              ON
               ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
               = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
            WHERE SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.21 CHECK_CONSTRAINTS view

Function

Identify the check constraints defined in this catalog that are owned by a given user.

Definition

```
CREATE VIEW CHECK_CONSTRAINTS
       AS SELECT
           CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME, CHECK_CLAUSE
         FROM DEFINITION_SCHEMA.CHECK_CONSTRAINTS
              JOIN
              DEFINITION_SCHEMA.SCHEMATA S
              ON
               ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
               = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
            WHERE SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.22 KEY_COLUMN_USAGE view

Function

Identify the columns defined in this catalog that are constrained as keys by a given user.

Definition

```
CREATE VIEW KEY_COLUMN_USAGE
       AS SELECT
           CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
           TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, ORDINAL_POSITION
         FROM DEFINITION_SCHEMA.KEY_COLUMN_USAGE
              JOIN
              DEFINITION_SCHEMA.SCHEMATA S
              ON
               ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
               = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
            WHERE SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.23 ASSERTIONS view

Function

Identify the assertions defined in this catalog that are owned by a given user.

Definition

```
CREATE VIEW ASSERTIONS
       AS SELECT
           CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
           IS_DEFERRABLE, INITIALLY_DEFERRED
         FROM DEFINITION_SCHEMA.ASSERTIONS
              JOIN
              DEFINITION_SCHEMA.SCHEMATA S
              ON
               ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
               = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
            WHERE SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.24 CHARACTER_SETS view

Function

Identify the character sets defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW CHARACTER_SETS
       AS SELECT
         CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
         FORM_OF_USE, NUMBER_OF_CHARACTERS,
         DEFAULT_COLLATE_CATALOG, DEFAULT_COLLATE_SCHEMA, DEFAULT_COLLATE_NAME
       FROM DEFINITION_SCHEMA.CHARACTER_SETS
       WHERE ( CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA,
               CHARACTER_SET_NAME, 'CHARACTER SET')
               IN
               ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                   FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
                   WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
           AND CHARACTER_SET_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.25 COLLATIONS view

Function

Identify the character collations defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW COLLATIONS
       AS SELECT
         COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME,
         CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
         PAD_ATTRIBUTE,
           COLLATION_TYPE, COLLATION_DEFINITION, COLLATION_DICTIONARY
       FROM DEFINITION_SCHEMA.COLLATIONS
       WHERE ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, 'COLLATION')
               IN
               ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                   FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
                   WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
           AND COLLATION_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not reference the COLLATIONS view.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

23.2.26 TRANSLATIONS view

Function

Identify the character translations defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW TRANSLATIONS
       AS SELECT
         TRANSLATION_CATALOG, TRANSLATION_SCHEMA, TRANSLATION_NAME,
         SOURCE_CHARACTER_SET_CATALOG, SOURCE_CHARACTER_SET_SCHEMA,
           SOURCE_CHARACTER_SET_NAME,
         TARGET_CHARACTER_SET_CATALOG, TARGET_CHARACTER_SET_SCHEMA,
           TARGET_CHARACTER_SET_NAME
       FROM DEFINITION_SCHEMA.TRANSLATIONS
       WHERE ( TRANSLATION_CATALOG, TRANSLATION_SCHEMA, TRANSLATION_NAME, 'TRANSLATION')

               IN
               ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                   FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
                   WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
           AND TRANSLATION_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not reference the TRANSLATIONS view.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

23.2.27 VIEW_TABLE_USAGE view

Function

Identify the tables on which viewed tables defined in this catalog and owned by a given user are dependent.

Definition

```
CREATE VIEW VIEW_TABLE_USAGE
       AS SELECT
         VIEW_CATALOG, VIEW_SCHEMA, VIEW_NAME,
         TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
       FROM DEFINITION_SCHEMA.VIEW_TABLE_USAGE
            JOIN
            DEFINITION_SCHEMA.SCHEMATA S
           ON
             ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
             ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE SCHEMA_OWNER = CURRENT_USER
           AND VIEW_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.28 VIEW_COLUMN_USAGE view

Function

Identify the columns on which viewed tables defined in this catalog and owned by a given user are dependent.

Definition

```
CREATE VIEW VIEW_COLUMN_USAGE AS
       SELECT VIEW_CATALOG, VIEW_SCHEMA, VIEW_NAME,
              TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
         FROM DEFINITION_SCHEMA.VIEW_COLUMN_USAGE
           JOIN
           DEFINITION_SCHEMA.SCHEMATA S
             ON
             ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
             ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE SCHEMA_OWNER = CURRENT_USER
           AND VIEW_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.29 CONSTRAINT_TABLE_USAGE view

Function

Identify the tables that are used by referential constraints, unique constraints, check constraints, and assertions defined in this catalog and owned by a given user.

Definition

```
CREATE VIEW CONSTRAINT_TABLE_USAGE
       AS
         SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
                CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME FROM
       (
         ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
                  CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
             FROM DEFINITION_SCHEMA.CHECK_COLUMN_USAGE )
       UNION
         ( SELECT PK.TABLE_CATALOG, PK.TABLE_SCHEMA, PK.TABLE_NAME,
                  FK.CONSTRAINT_CATALOG, FK.CONSTRAINT_SCHEMA, FK.CONSTRAINT_NAME
             FROM
             DEFINITION_SCHEMA.REFERENTIAL_CONSTRAINTS AS FK
             JOIN
             DEFINITION_SCHEMA.TABLE_CONSTRAINTS AS PK
               ON
                 ( FK.UNIQUE_CONSTRAINT_CATALOG, FK.UNIQUE_CONSTRAINT_SCHEMA,
                     FK.UNIQUE_CONSTRAINT_NAME )
               = ( PK.CONSTRAINT_CATALOG, PK.CONSTRAINT_SCHEMA, PK.CONSTRAINT_NAME )
         )
       )
       JOIN
         DEFINITION_SCHEMA.SCHEMATA S
            ON
             ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
             ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE S.SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.30 CONSTRAINT_COLUMN_USAGE view

Function

Identify the columns used by referential constraints, unique constraints, check constraints, and assertions defined in this catalog and owned by a given user.

Definition

```
CREATE VIEW CONSTRAINT_COLUMN_USAGE
       AS
         SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME,
                CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME FROM
           ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME,
                  CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
             FROM DEFINITION_SCHEMA.CHECK_COLUMN_USAGE )
           UNION
           ( SELECT K.TABLE_CATALOG, K.TABLE_SCHEMA, K.TABLE_NAME, K.COLUMN_NAME,
                    CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
               FROM DEFINITION_SCHEMA.TABLE_CONSTRAINTS
             JOIN DEFINITION_SCHEMA.KEY_COLUMN_USAGE AS K
               USING ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
         )
         JOIN
           DEFINITION_SCHEMA.SCHEMATA
            ON
             ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
             ( CATALOG_NAME, SCHEMA_NAME ) )
           WHERE SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.31 COLUMN_DOMAIN_USAGE view

Function

Identify the columns defined that are dependent on a domain defined in this catalog and owned by a user.

Definition

```
CREATE VIEW COLUMN_DOMAIN_USAGE
       AS SELECT
         D.DOMAIN_CATALOG, D.DOMAIN_SCHEMA, D.DOMAIN_NAME,
         TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
       FROM DEFINITION_SCHEMA.COLUMNS C
            JOIN
                DEFINITION_SCHEMA.DOMAINS D
                JOIN
                DEFINITION_SCHEMA.SCHEMATA S
                  ON ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA )
                  = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
            ON ( ( D.DOMAIN_CATALOG, D.DOMAIN_SCHEMA, D.DOMAIN_NAME )
                  = ( C.DOMAIN_CATALOG, C.DOMAIN_SCHEMA, C.DOMAIN_NAME ) )
         WHERE SCHEMA_OWNER = CURRENT_USER
           AND C.DOMAIN_NAME IS NOT NULL
           AND D.DOMAIN_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

ANSI Only–SQL3

23.2.32 ROUTINES view

Function

Identify the routines that are To Be Supplied.

Definition

```
CREATE VIEW ROUTINES
       AS SELECT
         ROUTINE_CATALOG, ROUTINE_SCHEMA, ROUTINE_NAME, To Be Supplied
       FROM DEFINITION_SCHEMA.ROUTINES
         JOIN DEFINITION_SCHEMA.SCHEMATA S
         ON
           ( ( ROUTINE_CATALOG, ROUTINE_SCHEMA )
           =
             ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE SCHEMA_OWNER = USER
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the ROUTINES view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

23.2.33 TRIGGERS view

Function

Identify the triggers that are owned by a given user.

Definition

```
CREATE VIEW TRIGGERS
       AS SELECT
         TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME, EVENT_MANIPULATION,
         EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE,
         EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE
```

ANSI Only--SQL3

```
         CONDITION_TIMING, CONDITION_REFERENCE_OLD_TABLE, CONDITION_REFERENCE_NEW_TABLE,

         COLUMN_LIST_IS_IMPLICIT
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
         CONDITION_TIMING, CONDITION_REFERENCE_OLD_TABLE, CONDITION_REFERENCE_NEW_TABLE,
```

```
       FROM DEFINITION_SCHEMA.TRIGGERS
         JOIN DEFINITION_SCHEMA.SCHEMATA S
         ON
           ( ( TRIGGER_CATALOG, TRIGGER_SCHEMA )
           =
             ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE SCHEMA_OWNER = USER
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the TRIGGERS view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

23.2.34 TRIGGERED_ACTIONS view

Function

Identify the triggered actions in a catalog that are owned by a given user.

Definition

```
CREATE VIEW TRIGGERED_ACTIONS
       AS SELECT
         TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME, ACTION_ORDER,
         ACTION_CONDITION, ACTION_STATEMENT_LIST, ACTION_ORIENTATION
       FROM DEFINITION_SCHEMA.TRIGGERED_ACTIONS
         JOIN DEFINITION_SCHEMA.SCHEMATA S
         ON
           ( ( TRIGGER_CATALOG, TRIGGER_SCHEMA )
           =
             ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE SCHEMA_OWNER = USER
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the TRIGGERED_ACTIONS view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

23.2.35 TRIGGERED_COLUMNS view

Function

For every schema owned by a user, identify the viewed tables that are dependent on a column that is described in a trigger.

Definition

```
CREATE VIEW TRIGGERED_COLUMNS
       AS SELECT
         TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME,
         EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE,
         EVENT_OBJECT_COLUMN,
       FROM DEFINITION_SCHEMA.TRIGGERED_COLUMNS
         JOIN DEFINITION_SCHEMA.SCHEMATA S
         ON
           ( ( TRIGGER_CATALOG, TRIGGER_SCHEMA )
           =
             ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE SCHEMA_OWNER = USER
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the TRIGGERED_COLUMNS view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

23.2.36 SQL_LANGUAGES view

Function

Identify the conformance levels, options, and dialects supported by the SQL-implementation processing data defined in this catalog.

Definition

```
CREATE VIEW SQL_LANGUAGES
       AS SELECT
         SQL_LANGUAGE_SOURCE, SQL_LANGUAGE_YEAR, SQL_LANGUAGE_CONFORMANCE,
           SQL_LANGUAGE_INTEGRITY, SQL_LANGUAGE_IMPLEMENTATION,
           SQL_LANGUAGE_BINDING_STYLE, SQL_LANGUAGE_PROGRAMMING_LANGUAGE
       FROM DEFINITION_SCHEMA.SQL_LANGUAGES
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

23.2.37 SQL_IDENTIFIER domain

Function

Define a domain that contains all valid <identifier>s

Definition

```
CREATE DOMAIN SQL_IDENTIFIER AS CHARACTER VARYING (L)
       CHARACTER SET SQL_TEXT
```

Description

1) This domain specifies any variable-length character value that conforms to the rules for an SQL <identifier>.

   Note: There is no way in SQL to specify a <domain constraint> that would be true for any valid SQL <identifier> and false for all other values.

2) *L* is the implementation-defined maximum length of <identifier>.

23.2.38 CHARACTER_DATA domain

Function

Define a domain that contains any character data.

Definition

```
CREATE DOMAIN CHARACTER_DATA AS CHARACTER VARYING (ML)
       CHARACTER SET SQL_TEXT
```

Description

1) This domain specifies any character data.

2) *ML* is the implementation-defined maximum length of a variable-length character string.

23.2.39 CARDINAL_NUMBER domain

Function

Define a domain that contains a non-negative number.

Definition

```
CREATE DOMAIN CARDINAL_NUMBER AS INTEGER
       CONSTRAINT CARDINAL_NUMBER_DOMAIN_CHECK CHECK ( VALUE >= 0 )
```

Description

1) The domain CARDINAL_NUMBER contains any non-negative number that is less than the implementation-defined maximum for INTEGER (i.e., the implementation-defined value of NUMERIC_PRECISION_RADIX raised to the power of implementation-defined NUMERIC_PRECISION).

23.3 Definition Schema

23.3.1 Introduction

The base tables and views of the Definition Schema are all defined in a <schema definition> for the schema named named DEFINITION_SCHEMA. The table definitions are as as complete as the definitional power of SQL allows. The table definitions are supplemented with assertions where appropriate; see Subclause 23.4, "Assertions on the base tables". Each description comprises three parts:

1) The function of the definition is stated.

2) The SQL definition of the object is presented as a <table definition> or <view definition>.

3) An explanation of the object.

The specification provides only a model of the base tables and views that are required, and does not imply that an implementation shall provide the functionality in the manner described in this clause.

An instance of a definition schema describes an instance of a cluster of catalogs (see Subclause 4.22, "Clusters of catalogs").

23.3.2 DEFINITION_SCHEMA Schema

Function

Create the schema that is to contain the base tables that underlie the Information Schema

Definition

```
CREATE SCHEMA DEFINITION_SCHEMA
       AUTHORIZATION DEFINITION_SCHEMA
```

Description

None.

23.3.3 USERS base table

Function

The USERS table has one row for each <authorization identifier> referenced in the Information Schema. These are all those <authorization identifier>s that may grant or receive privileges as well as those that may create a schema, or currently own a schema created through a <schema definition>.

Definition

```
CREATE TABLE USERS
       (
       USER_NAME     INFORMATION_SCHEMA.SQL_IDENTIFIER
           CONSTRAINT USERS_PRIMARY_KEY PRIMARY KEY
         CONSTRAINT USERS_CHECK CHECK ( ROLE_NAME NOT IN
                     ( SELECT USER_NAME FROM ROLES ) )
       )
```

Description

1) The values of USER_NAME are <authorization identifier>s that are known.

23.3.4 ROLES base table

Function

The ROLES table has one row for each <role name> for each role known to the database management system.

Definition

```
CREATE TABLE ROLES
       (
       ROLE_NAME              INFORMATION_SCHEMA.SQL_IDENTIFIER       PRIMARY KEY,
         CONSTRAINT ROLES_CHECK CHECK ( ROLE_NAME NOT IN
                     ( SELECT USER_NAME FROM USERS ) )
       )
```

Description

1) A row is inserted into this table each time a <role definition> is executed. A row is deleted from this table each time the <drop role statement> is executed.

2) The value of ROLE_NAME is the <role name> defined by <role definition>.

23.3.5 SCHEMATA base table

Function

The SCHEMATA table has one row for each schema.

Definition

```
CREATE TABLE SCHEMATA
       (
       CATALOG_NAME                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
       SCHEMA_NAME                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       SCHEMA_OWNER                   INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT SCHEMA_OWNER_NOT_NULL NOT NULL,
       DEFAULT_CHARACTER_SET_CATALOG  INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DEFAULT_CHARACTER_SET_SCHEMA   INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DEFAULT_CHARACTER_SET_NAME     INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT SCHEMATA_PRIMARY_KEY PRIMARY KEY ( CATALOG_NAME, SCHEMA_NAME ),

       CONSTRAINT SCHEMATA_FOREIGN_KEY FOREIGN KEY ( SCHEMA_OWNER )
         REFERENCES USERS
       )
```

Description

1) All the values of CATALOG_NAME are the name of the catalog in which the schemata are included.

2) The values of SCHEMA_NAME are the unqualified schema names of the schemata in the catalog.

3) The values of SCHEMA_OWNER are the authorization identifiers that own the schemata.

4) The values of DEFAULT_CHARACTER_SET_CATALOG, DEFAULT_CHARACTER_SET_SCHEMA, and DEFAULT_CHARACTER_SET_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the default character set for columns and domains in the schemata.

23.3.6 ABSTRACT_DATA_TYPES base table

Function

The ABSTRACT_DATA_TYPES table has one row for each abstract data type.

Definition

```
CREATE TABLE ABSTRACT_DATA_TYPES
       (
         ABSTRACT_DATA_TYPE_CATALOG              INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_SCHEMA               INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_NAME                 INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_EQUALS               INFORMATION_SCHEMA.CHARACTER_DATA,
         ABSTRACT_DATA_TYPE_LESS_THAN            INFORMATION_SCHEMA.CHARACTER_DATA,

  CONSTRAINT ABSTRACT_DATA_TYPES_PRIMARY_KEY
           PRIMARY KEY (ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                        ABSTRACT_DATA_TYPE_NAME, ABSTRACT_DATA_TYPE_LANGUAGE),

  CONSTRAINT ABSTRACT_DATA_TYPES_FOREIGN_KEY
           FOREIGN KEY (ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA)
             REFERENCES SCHEMATA
       )
```

Description

1) A row is inserted in this table whenever an <abstract data type definition> is executed and a row is deleted whenever a <drop data type statement> is executed.

2) The values of ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA and ABSTRACT_DATA_TYPE_NAME are the qualified name of the abstract data type that is defined.

3) The value of ABSTRACT_DATA_TYPE_LANGUAGE is one of ADA, C, COBOL, FORTRAN, PASCAL, PLI, as specified in the <language clause> of the associated <user data type definition>.

4) The values of ABSTRACT_DATA_TYPE_EQUALS and ABSTRACT_DATA_TYPE_LESS_THAN are respectively the <external function name>s specified in the <comparisons clause> of the associated <user data type definition>.

23.3.7 DATA_TYPE_DESCRIPTOR base table

Function

The DATA_TYPE_DESCRIPTOR table has one row for each domain and one row for each column (in each table) that is defined as having a data type rather than a domain. It effectively contains a representation of the data type descriptors.

Definition

```
CREATE TABLE DATA_TYPE_DESCRIPTOR
       (
       TABLE_OR_DOMAIN_CATALOG             INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_OR_DOMAIN_SCHEMA              INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_OR_DOMAIN_NAME                INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_NAME                         INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DATA_TYPE                           INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_OR_DOMAIN_DATA_TYPE_NOT_NULL NOT NULL,
       CHARACTER_MAXIMUM_LENGTH            INFORMATION_SCHEMA.CARDINAL_NUMBER,
       CHARACTER_OCTET_LENGTH              INFORMATION_SCHEMA.CARDINAL_NUMBER,
       COLLATION_CATALOG                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLLATION_SCHEMA                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLLATION_NAME                      INFORMATION_SCHEMA.SQL_IDENTIFIER,
       NUMERIC_PRECISION                   INFORMATION_SCHEMA.CARDINAL_NUMBER,
       NUMERIC_PRECISION_RADIX             INFORMATION_SCHEMA.CARDINAL_NUMBER,
       NUMERIC_SCALE                       INFORMATION_SCHEMA.CARDINAL_NUMBER,
       DATETIME_PRECISION                  INFORMATION_SCHEMA.CARDINAL_NUMBER,
```

ANSI Only--SQL3

```
       INTERVAL_CODE                       INFORMATION_SCHEMA.CHARACTER_DATA,
       INTERVAL_PRECISION                  INFORMATION_SCHEMA.CARDINAL_NUMBER,
```

```
       ABSTRACT_DATA_TYPE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       ABSTRACT_DATA_TYPE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       ABSTRACT_DATA_TYPE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT TABLE_OR_DOMAIN_CHECK_COMBINATIONS
         CHECK ( DATA_TYPE IN ( 'CHARACTER', 'CHARACTER VARYING',
                                'BIT', 'BIT VARYING' )
               AND ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
                   COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                     IS NOT NULL
               AND ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
                     NUMERIC_SCALE, DATETIME_PRECISION,
                     ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                     ABSTRACT_DATA_TYPE_NAME ) IS NULL
```

ANSI Only--SQL3

```
               AND ( INTERVAL_CODE, INTERVAL_PRECISION )
                     IS NULL
```

```
         OR
               DATA_TYPE IN ( 'REAL', 'DOUBLE PRECISION', 'FLOAT' )
               AND ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
                   COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                       IS NULL
               AND NUMERIC_PRECISION IS NOT NULL
```

```
        AND NUMERIC_PRECISION_RADIX = 2
        AND NUMERIC_SCALE IS NULL
        AND DATETIME_PRECISION IS NULL
        AND ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                                ABSTRACT_DATA_TYPE_NAME ) IS NULL
```

ANSI Only--SQL3

```
        AND ( INTERVAL_CODE, INTERVAL_PRECISION )
              IS NULL
```

```
  OR
        DATA_TYPE IN ( 'INTEGER', 'SMALLINT', 'NUMERIC', 'DECIMAL' )
        AND ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
            COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                IS NULL
```

ANSI Only--SQL3

```
        AND ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX ) IS NOT NULL
```

```
        AND NUMERIC_SCALE IS NOT NULL
        AND ( NUMERIC_SCALE <> 0 AND NUMERIC_PRECISION_RADIX = 10
              OR
              NUMERIC_SCALE = 0 AND NUMERIC_PRECISION_RADIX IN ( 2, 10 ) )
        AND DATETIME_PRECISION IS NULL
        AND ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                                ABSTRACT_DATA_TYPE_NAME ) IS NULL
```

ANSI Only--SQL3

```
        AND ( INTERVAL_CODE, INTERVAL_PRECISION )
              IS NULL
```

```
  OR
        DATA_TYPE IN ( 'DATE', 'TIME', 'TIMESTAMP,
                       'TIME WITH TIME ZONE', 'TIMESTAMP WITH TIME ZONE' )
        AND ( CHARACTER_MAXIMUM_LENGTH,
            CHARACTER_OCTET_LENGTH,
            COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                IS NULL
```

ANSI Only--SQL3

```
        AND ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX ) IS NOT NULL
```

```
        AND NUMERIC_SCALE IS NULL
        AND DATETIME_PRECISION IS NOT NULL
        AND ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                                ABSTRACT_DATA_TYPE_NAME ) IS NULL
```

ANSI Only--SQL3

```
        AND ( INTERVAL_CODE, INTERVAL_PRECISION )
              IS NULL
```

```
  OR
        DATA_TYPE = 'INTERVAL'
        AND ( CHARACTER_MAXIMUM_LENGTH,
            CHARACTER_OCTET_LENGTH,
            COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                IS NULL
```

ANSI Only--SQL3

```
        AND ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX ) IS NOT NULL
```

```
        AND NUMERIC_SCALE IS NULL
        AND DATETIME_PRECISION IS NOT NULL
        AND ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                                ABSTRACT_DATA_TYPE_NAME ) IS NULL
        AND INTERVAL_CODE IN
                                ( 'YEAR', 'MONTH', 'DAY', 'HOUR',
                                  'MINUTE', 'SECOND', 'YEAR TO MONTH',
                                  'DAY TO HOUR', 'DAY TO MINUTE',
                                  'DAY TO SECOND', 'HOUR TO MINUTE',
                                  ''HOUR TO SECOND', 'MINUTE TO SECOND' )
        AND INTERVAL_PRECISION
          IS NOT NULL
  OR
        DATA_TYPE = 'BOOLEAN'
        AND ( CHARACTER_MAXIMUM_LENGTH,
            CHARACTER_OCTET_LENGTH,
            COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                IS NULL
        AND ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX ) IS NULL
        AND NUMERIC_SCALE IS NULL
        AND DATETIME_PRECISION IS NULL
        AND ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                                ABSTRACT_DATA_TYPE_NAME ) IS NULL
```

ANSI Only--SQL3

```
        AND ( INTERVAL_CODE, INTERVAL_PRECISION )
              IS NULL
```

```
  OR
        DATA_TYPE = 'USER_DEFINED'
        AND ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX )
              NUMERIC_SCALE, DATETIME_PRECISION,
            CHARACTER_OCTET_LENGTH,
              CHARACTER_MAXIMUM_LENGTH ) IS NULL
```

ANSI Only--SQL3

```
        AND ( INTERVAL_CODE, INTERVAL_PRECISION )
              IS NULL
```

```
  ),

CONSTRAINT DATA_TYPE_DESCRIPTOR_PRIMARY_KEY
  PRIMARY KEY ( TABLE_OR_DOMAIN_CATALOG, TABLE_OR_DOMAIN_SCHEMA,
                  TABLE_OR_DOMAIN_NAME, COLUMN_NAME ),
```

```
CONSTRAINT DATA_TYPE_CHECK_REFERENCES_COLLATION
    CHECK ( COLLATION_CATALOG
            <> ANY ( SELECT CATALOG_NAME FROM SCH
       OR
           ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ) IN<REFERE
           ( SELECT COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME
             FROM COLLATIONS ) ),

CONSTRAINT DATA_TYPE_DESCRIPTOR_FOREIGN_KEY_SCHEMATA
  FOREIGN KEY ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA)
  REFERENCES SCHEMATA,

CONSTRAINT DATA_TYPE_DESCRIPTOR_CHECK_USED
  CHECK ( (TABLE_OR_DOMAIN_CATALOG, TABLE_OR_DOMAIN_SCHEMA,
             TABLE_OR_DOMAIN_NAME, COLUMN_NAME)
    IN (
        SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
            FROM COLUMNS
        UNION
        SELECT DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME, ''
            FROM DOMAINS )
      )
)
```

Description

1) The values of TABLE_OR_DOMAIN_CATALOG and TABLE_OR_DOMAIN_SCHEMA are the catalog name and the unqualified schema name, respectively, of the schema that contains the object (domain or column) to which the data type descriptor belongs.

2) Case:

   a) If the length of COLUMN_NAME is 0, then the value of TABLE_OR_DOMAIN_NAME is the name of the domain to which the data type descriptor belongs.

   b) Otherwise, TABLE_OR_DOMAIN_NAME is the name of the table and COLUMN_NAME is the name of the column in that table to which the data type descriptor belongs.

3) The values of DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH, COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE, DATETIME_PRECISION, ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA, and ABSTRACT_DATA_TYPE_NAME contain the data type of the domain or column being defined, the maximum length in characters or bits of the column if it is a character or bit type respectively, maximum length in octets of the column if it is a character type, the qualified name of the applicable collation if it is a character type, the precision and radix of the precision if it is a numeric type,
ANSI the scale if it is a numeric type,
the precision if it is a datetime or interval type, and the qualified name of the abstract data type, if specified.

ANSI Only–SQL3

4) If DATA_TYPE is 'INTERVAL', then the values of INTERVAL_CODE are the value for <interval qualifier> (as specified in Table 27, "Codes used for <interval qualifier>s in Dynamic SQL") for the data type being described; otherwise, INTERVAL_CODE is the general null value.

5) If DATA_TYPE is 'INTERVAL', then the values of INTERVAL_PRECISION are the interval leading field precision of the data type being described; otherwise, INTERVAL_PRECISION is the general null value.

---

6) The values of ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA, and ABSTRACT_DATA_TYPE_NAME are null if the column being is not defined as an abstract data type. Otherwise, the values of ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA, and ABSTRACT_DATA_TYPE_NAME are the qualified name of the abstract data type used by the column being described.

23.3.8 DOMAINS base table

Function

The DOMAINS table has one row for each domain. It effectively contains a representation of the domain descriptors.

Definition

```
CREATE TABLE DOMAINS
       (
       DOMAIN_CATALOG        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DOMAIN_SCHEMA         INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DOMAIN_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DOMAIN_DEFAULT        INFORMATION_SCHEMA.CHARACTER_DATA,
       NULL_CLASS_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       NULL_CLASS_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       NULL_CLASS_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT DOMAINS_PRIMARY_KEY
         PRIMARY KEY ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME ),

       CONSTRAINT DOMAINS_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( DOMAIN_CATALOG, DOMAIN_SCHEMA ) REFERENCES SCHEMATA,

       CONSTRAINT DOMAINS_FOREIGN_KEY_NULL_CLASSES
         FOREIGN KEY ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME )
           REFERENCES NULL_CLASSES,

       CONSTRAINT DOMAIN_CHECK_DATA_TYPE
         CHECK ( DOMAIN_CATALOG
                   <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
             OR
                 ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME, '' ) IN
                 ( SELECT TABLE_OR_DOMAIN_CATALOG, TABLE_OR_DOMAIN_SCHEMA,
                          TABLE_OR_DOMAIN_NAME, COLUMN_NAME
                   FROM DATA_TYPE_DESCRIPTOR ) ),
       )
```

Description

1) The values of DOMAIN_CATALOG and DOMAIN_SCHEMA are the catalog name and unqualified schema name, respectively, of the schema in which the domain is defined.

2) The value of DOMAIN_NAME is the name of the domain.

3) The value of CONSTRAINT_NAME is the name of the domain constraint, if any, for this domain.

4) The value of DOMAIN_DEFAULT is null if the domain being described has no default value. Otherwise, the value of DOMAIN_DEFAULT is a character representation of the default value for the domain that obeys the rules specified for <default option> in Subclause 11.5, "<default clause>".

5) The values of IS_DEFERRABLE have the following meanings:

YES	The domain constraint is deferrable.
NO	The domain constraint is not deferrable.

6) The values of INITIALLY_DEFERRED have the following meanings:

YES	The domain constraint is initially deferred.
NO	The domain constraint is initially immediate.

7) The values of NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, and NULL_CLASS_NAME are the name of the null class of the domain.

23.3.9 DOMAIN_CONSTRAINTS base table

Function

The DOMAIN_CONSTRAINTS table has one row for each domain constraint associated with a domain. It effectively contains a representation of the domain constraint descriptors.

Definition

```
CREATE TABLE DOMAIN_CONSTRAINTS
       (
       CONSTRAINT_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DOMAIN_CATALOG        INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT DOMAIN_CATALOG_NOT_NULL NOT NULL,
       DOMAIN_SCHEMA         INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT DOMAIN_SCHEMA_NOT_NULL NOT NULL,
       DOMAIN_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT DOMAIN_NAME_NOT_NULL NOT NULL,
       IS_DEFERRABLE         INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT DOMAIN_CONSTRAINTS_DEFERRABLE_NOT_NULL NOT NULL,
       INITIALLY_DEFERRED    INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT DOMAIN_CONSTRAINTS_INITIALLY_DEFERRED_NOT_NULL NOT NULL,
       CONSTRAINT DOMAIN_CONSTRAINTS_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),
       CONSTRAINT DOMAIN_CONSTRAINTS_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT DOMAIN_CONSTRAINTS_FOREIGN_KEY_CHECK_CONSTRAINTS
         FOREIGN KEY ( DOMAIN_CATALOG, DOMAIN_SCHEMA, CONSTRAINT_NAME )
           REFERENCES CHECK_CONSTRAINTS,

       CONSTRAINT DOMAIN_CONSTRAINTS_FOREIGN_KEY_DOMAINS
         FOREIGN KEY ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME )
           REFERENCES DOMAINS,

       CONSTRAINT DOMAIN_CONSTRAINTS_CHECK_DEFERRABLE
         CHECK ( ( IS_DEFERRABLE, INITIALLY_DEFERRED ) IN
                 ( VALUES ( 'NO',  'NO' ),
                          ( 'YES', 'NO' ),
                          ( 'YES', 'YES' ) ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG and CONSTRAINT_SCHEMA are the catalog name and unqualified schema name of the schema in which the domain constraint is defined.

2) The value of CONSTRAINT_NAME is the name of the domain constraint.

3) The values of DOMAIN_CATALOG, DOMAIN_SCHEMA and DOMAIN_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the domain in which the domain constraint is defined.

4) The values of IS_DEFERRABLE have the following meanings:

YES	The domain constraint is deferrable.
NO	The domain constraint is not deferrable.

5) The values of INITIALLY_DEFERRED have the following meanings:

YES	The domain constraint is initially deferred.
NO	The domain constraint is initially immediate.

23.3.10 TABLES base table

Function

The TABLES table contains one row for each table including views. It effectively contains a representation of the table descriptors.

Definition

```
CREATE TABLE TABLES
       (
       TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_TYPE             INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_TYPE_NOT_NULL NOT NULL,
         CONSTRAINT TABLE_TYPE_CHECK CHECK ( TABLE_TYPE IN
                ( 'BASE TABLE', 'VIEW', 'GLOBAL TEMPORARY', 'LOCAL TEMPORARY' ) ),
         CONSTRAINT CHECK_TABLE_IN_COLUMNS
           CHECK ( ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
                ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                    FROM COLUMNS ) ),

       CONSTRAINT TABLES_PRIMARY_KEY
         PRIMARY KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ),

       CONSTRAINT TABLES_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT TABLES_CHECK_NOT_VIEW CHECK ( NOT EXISTS
         ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
             FROM TABLES
             WHERE TABLE_TYPE = 'VIEW'
           EXCEPT
           SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
             FROM VIEWS ) )
       )
```

Description

1) The values of TABLE_CATALOG and TABLE_SCHEMA are the catalog name and unqualified schema name, respectively, of the schema in which the table is defined.

2) The value of TABLE_NAME is the name of the table.

3) The values of TABLE_TYPE have the following meanings:

BASE TABLE	The table being described is a persistent base table.
VIEW	The table being described is a viewed table.
GLOBAL TEMPORARY	The table being described is a global temporary table.
LOCAL TEMPORARY	The table being described is a created local temporary table.

23.3.11 VIEWS base table

Function

The VIEWS table contains one row for each row in the TABLES table with a TABLE_TYPE of 'VIEW'. Each row describes the query expression that defines a view. The table effectively contains a representation of the view descriptors.

Definition

```
CREATE TABLE VIEWS
       (
       TABLE_CATALOG   INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME      INFORMATION_SCHEMA.SQL_IDENTIFIER,
       VIEW_DEFINITION INFORMATION_SCHEMA.CHARACTER_DATA,
       CHECK_OPTION    INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT CHECK_OPTION_NOT_NULL NOT NU
         CONSTRAINT CHECK_OPTION_CHECK
           CHECK ( CHECK_OPTION IN ( 'CASCADED', 'LOCAL', 'NONE' ) ),
       IS_UPDATABLE       INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT IS_UPDATABLE_NOT_NULL NOT NU
         CONSTRAINT IS_UPDATABLE_CHECK
           CHECK ( IS_UPDATABLE IN ( 'YES', 'NO' ) ),

       CONSTRAINT VIEWS_PRIMARY_KEY
         PRIMARY KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ),

       CONSTRAINT VIEWS_IN_TABLES_CHECK
         CHECK ( ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
         ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
             FROM TABLES
             WHERE TABLE_TYPE = 'VIEW' ) ),

       CONSTRAINT VIEWS_IS_UPDATABLE_CHECK_OPTION_CHECK
         CHECK ( ( IS_UPDATABLE, CHECK_OPTION ) NOT IN
           ( VALUES ( 'NO', 'CASCADED' ), ( 'NO', 'LOCAL' ) ) )
       )
```

Description

1) The values of TABLE_CATALOG and TABLE_SCHEMA are the catalog name and unqualified schema name, respectively, of the schema in which the viewed table is defined.

2) The value of TABLE_NAME is the name of the viewed table.

3) Case:

   a) If the character representation of the <query expression> contained in the <view definition> that defined the view being described can be represented without truncation, then the value of VIEW_DEFINITION is that character representation.

   b) Otherwise, the value of VIEW_DEFINITION is the null value.

   Note: Any implicit <column reference>s that were contained in the <query expression> associated with the <view definition> are replaced by explicit <column reference>s in VIEW_DEFINITION.

4) The values of CHECK_OPTION have the following meanings:

CASCADED	The <view definition> contains WITH CASCADED CHECK OPTION.
LOCAL	The <view definition> contains WITH LOCAL CHECK OPTION.
NONE	The <view definition> does not contain WITH CHECK OPTION.

5) The values of IS_UPDATABLE have the following meanings:

YES	The <view definition> simply contains a <query expression> that is updatable.
NO	The <view definition> simply contains a <query expression> that is not updatable.

23.3.12 COLUMNS base table

Function

The COLUMNS table has one row for each non-inherited column. It effectively contains a representation of the column descriptors.

Definition

```
CREATE TABLE COLUMNS
       (
       TABLE_CATALOG         INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       ORDINAL_POSITION      INFORMATION_SCHEMA.CARDINAL_NUMBER
         CONSTRAINT COLUMN_POSITION_NOT_NULL NOTNULL,
       DOMAIN_CATALOG        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DOMAIN_SCHEMA         INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DOMAIN_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_DEFAULT        INFORMATION_SCHEMA.CHARACTER_DATA,
       IS_NULLABLE           INFORMATION_SCHEMA.CHARACTER_DATA,
       NULL_CLASS_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       NULL_CLASS_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       NULL_CLASS_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT COLUMNS_PRIMARY_KEY
         PRIMARY KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ),

       CONSTRAINT COLUMNS_UNIQUE
         UNIQUE ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, ORDINAL_POSITION ),

       CONSTRAINT COLUMNS_FOREIGN_KEY_TABLES
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
           REFERENCES TABLES,

       CONSTRAINT COLUMNS_FOREIGN_KEY_NULL_CLASSES
         FOREIGN KEY ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME )
           REFERENCES NULL_CLASSES,

       CONSTRAINT COLUMNS_CHECK_REFERENCES_DOMAIN
         CHECK ( DOMAIN_CATALOG
                   <> ANY ( SELECT  CATALOG_NAME FROM SCHEMATA )
             OR
                ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME ) IN
                ( SELECT DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME
                  FROM DOMAINS ) ),

       CONSTRAINT COLUMN_CHECK_DATA_TYPE
         CHECK ( DOMAIN_CATALOG
                   <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
             OR
            ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME ) IS NOT NULL
                AND
              ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ) NOT IN
              ( SELECT TABLE_OR_DOMAIN_CATALOG, TABLE_OR_DOMAIN_SCHEMA,
                         TABLE_OR_DOMAIN_NAME, COLUMN_NAME
                  FROM DATA_TYPE_DESCRIPTOR )
                AND
                   ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME ) IS NULL
             OR
```

```
        ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME ) IS NULL
          AND
        ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ) IN
        ( SELECT TABLE_OR_DOMAIN_CATALOG, TABLE_OR_DOMAIN_SCHEMA,
                  TABLE_OR_DOMAIN_NAME, COLUMN_NAME
            FROM DATA_TYPE_DESCRIPTOR )
          AND
            ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME ) IS NOT NULL
    ) )
)
```

Description

1) Case:

   a) If a column is described by a column descriptor included in a table descriptor, then the table descriptor and the column descriptor are associated with that column.

   b) If a column is described by a column descriptor included in a view descriptor, then the view descriptor and the corresponding column descriptor of the table of the <query expression> are associated with that column.

2) The values of TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the table containing the column being described.

3) The value of COLUMN_NAME is the name of the column being described.

4) The values of DOMAIN_CATALOG, DOMAIN_SCHEMA, and DOMAIN_NAME are null if the column being described is not defined using a <domain name>. Otherwise, the values of DOMAIN_CATALOG, DOMAIN_SCHEMA, and DOMAIN_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the domain used by the column being described.

5) The value of NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, and NULL_CLASS_NAME is the name of the null class of the column.

6) The value of ORDINAL_POSITION is the ordinal position of the column in the table.

7) The value of COLUMN_DEFAULT is null if the column being described has no default value or if its default value comes only from a domain. Otherwise, the value of COLUMN_DEFAULT is a character representation of the default value for the column that obeys the rules specified for <default option> in Subclause 11.5, "<default clause>".

8) The values of IS_NULLABLE have the following meanings:

YES	The columns is possibly nullable.
NO	The column is known not nullable.

23.3.13 VIEW_TABLE_USAGE base table

Function

The VIEW_TABLE_USAGE table has one row for each table referenced in the <query expression> of a view.

Definition

```
CREATE TABLE VIEW_TABLE_USAGE
       (
       VIEW_CATALOG           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       VIEW_SCHEMA            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       VIEW_NAME              INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT VIEW_TABLE_USAGE_PRIMARY_KEY
       PRIMARY KEY ( VIEW_CATALOG, VIEW_SCHEMA, VIEW_NAME,
                     TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ),

       CONSTRAINT VIEW_TABLE_USAGE_CHECK_REFERENCES_TABLES
           CHECK ( TABLE_CATALOG
                     <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
               OR
                   ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
                   ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                     FROM TABLES ) ),

       CONSTRAINT VIEW_TABLE_USAGE_FOREIGN_KEY_VIEWS
       FOREIGN KEY ( VIEW_CATALOG, VIEW_SCHEMA, VIEW_NAME )
         REFERENCES VIEWS
       )
```

Description

1) The values of VIEW_CATALOG, VIEW_SCHEMA, and VIEW_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the view being described.

2) The values of TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of a table that the view requires.

23.3.14 VIEW_COLUMN_USAGE base table

Function

The VIEW_COLUMN_USAGE table has one row for each column referenced by a view.

Definition

```
CREATE TABLE VIEW_COLUMN_USAGE
       (
       VIEW_CATALOG           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       VIEW_SCHEMA            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       VIEW_NAME              INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT VIEW_COLUMN_USAGE_PRIMARY_KEY
       PRIMARY KEY ( VIEW_CATALOG, VIEW_SCHEMA, VIEW_NAME,
                     TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ),

       CONSTRAINT VIEW_COLUMN_USAGE_CHECK_REFERENCES_COLUMNS
           CHECK ( TABLE_CATALOG
                   <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
              OR
                  ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ) IN
                  ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
                    FROM COLUMNS ) ),

       CONSTRAINT VIEW_COLUMN_USAGE_FOREIGN_KEY_VIEWS
       FOREIGN KEY ( VIEW_CATALOG, VIEW_SCHEMA, VIEW_NAME )
         REFERENCES VIEWS
       )
```

Description

1) The values of VIEW_CATALOG, VIEW_SCHEMA, and VIEW_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the view being described.

2) The values of TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME are the catalog name, unqualified schema name, qualified identifier, and identifier, respectively, of a column that is explicitly or implicitly referenced in the <query expression> of the view being described.

23.3.15 TABLE_CONSTRAINTS base table

Function

The TABLE_CONSTRAINTS table has one row for each table constraint associated with a table. It effectively contains a representation of the table constraint descriptors.

Definition

```
CREATE TABLE TABLE_CONSTRAINTS
       (
       CONSTRAINT_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_TYPE       INFORMATION_SCHEMA.CHARACTER_DATA
       CONSTRAINT CONSTRAINT_TYPE_NOT_NULL NOT NULL
         CONSTRAINT CONSTRAINT_TYPE_CHECK

           CHECK ( CONSTRAINT_TYPE IN
                   ( 'UNIQUE',
                     'PRIMARY KEY',
                     'FOREIGN KEY',
                     'CHECK' ) ),

       TABLE_CATALOG         INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TABLE_CONSTRAINTS_TABLE_CATALOG_NOT_NULL NOT NULL,
       TABLE_SCHEMA          INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TABLE_CONSTRAINTS_TABLE_SCHEMA_NOT_NULL NOT NULL,
       TABLE_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TABLE_CONSTRAINTS_TABLE_NAME_NOT_NULL NOT NULL,
       IS_DEFERRABLE INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_CONSTRAINTS_IS_DEFERRABLE_NOT_NULL NOT NULL,
       INITIALLY_DEFERRED    INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_CONSTRAINTS_INITIALLY_DEFERRED_NOT_NULL

       CONSTRAINT TABLE_CONSTRAINTS_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),

       CONSTRAINT TABLE_CONSTRAINTS_CHECK_REFERENCES_TABLES
         CHECK ( TABLE_CATALOG
                 <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
             OR
                ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
                ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                  FROM TABLES ) ),

       CONSTRAINT TABLE_CONSTRAINTS_DEFERRED_CHECK
         CHECK ( ( IS_DEFERRABLE, INITIALLY_DEFERRED ) IN
                 ( VALUES ( 'NO',  'NO' ),
                          ( 'YES', 'NO' ),
                          ( 'YES', 'YES' ) ) ),

       CONSTRAINT TABLE_CONSTRAINTS_CHECK_VIEWS
         CHECK ( TABLE_CATALOG
                 <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
             OR
            ( ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
              ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                FROM TABLES
                WHERE TABLE_TYPE <> 'VIEW' ) ),
```

```
CONSTRAINT TABLE_CONSTRAINTS_UNIQUE_CHECK
  CHECK ( 1 =
    ( SELECT COUNT (*)
      FROM ( SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
             FROM TABLE_CONSTRAINTS
             WHERE CONSTRAINT_TYPE IN
                     ( 'UNIQUE', 'PRIMARY KEY' )
      UNION ALL
      SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
      FROM REFERENTIAL_CONSTRAINTS
      UNION ALL
      SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
      FROM CHECK_CONSTRAINTS ) AS X
      WHERE ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
          = ( X.CONSTRAINT_CATALOG, X.CONSTRAINT_SCHEMA, X.CONSTRAINT_NAME )
    ) ),

CONSTRAINT UNIQUE_TABLE_PRIMARY_KEY_CHECK
  CHECK ( UNIQUE ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                   FROM TABLE_CONSTRAINTS
                   WHERE CONSTRAINT_TYPE = 'PRIMARY KEY' ) )
)
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the constraint being described. If the <table constraint definition> or <add table constraint definition> that defined the constraint did not specify a <constraint name>, then the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are implementation-defined.

2) The values of CONSTRAINT_TYPE have the following meanings:

FOREIGN KEY	The constraint being described is a foreign key constraint.
UNIQUE	The constraint being described is a unique constraint.
PRIMARY KEY	The constraint being described is a primary key constraint.
CHECK	The constraint being described is a check constraint.

3) The values of TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME are the catalog name, the unqualified schema name, and the qualified identifier of the name of the table to which the table constraint being described applies.

4) The values of IS_DEFERRABLE have the following meanings:

YES	The table constraint is deferrable.
NO	The table constraint is not deferrable.

5) The values of INITIALLY_DEFERRED have the following meanings:

YES	The table constraint is initially deferred.
NO	The table constraint is initially immediate.

23.3.16 KEY_COLUMN_USAGE base table

Function

The KEY_COLUMN_USAGE table has one or more rows for each row in the TABLE_CONSTRAINTS table that has a CONSTRAINT_TYPE of ‘‘UNIQUE’’, ‘‘PRIMARY KEY’’, or ‘‘FOREIGN KEY’’. The rows list the columns that constitute each unique constraint, and the referencing columns in each foreign key constraint.

Definition

```
CREATE TABLE KEY_COLUMN_USAGE
       (
       CONSTRAINT_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG         INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT KEY_COLUMN_TABLE_CATALOG_NOT_NULL NOT NULL,
       TABLE_SCHEMA          INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT KEY_COLUMN_TABLE_SCHEMA_NOT_NULL NOT NULL,
       TABLE_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT KEY_COLUMN_TABLE_NAME_NOT_NULL NOT NULL,
       COLUMN_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       ORDINAL_POSITION      INFORMATION_SCHEMA.CARDINAL_NUMBER
         CONSTRAINT KEY_COLUMN_ORDINAL_POSITION_NOT_NULL NOT NULL,

       CONSTRAINT KEY_COLUMN_USAGE_PRIMARY_KEY
       PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
                       COLUMN_NAME ),

       CONSTRAINT KEY_COLUMN_USAGE_UNIQUE
       UNIQUE ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA,
                CONSTRAINT_NAME, ORDINAL_POSITION ),

       CONSTRAINT KEY_COLUMN_USAGE_FOREIGN_KEY_COLUMNS
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME )
           REFERENCES COLUMNS,

       CONSTRAINT KEY_COLUMN_CONSTRAINT_TYPE_CHECK
       CHECK ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
               IN ( SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                      FROM TABLE_CONSTRAINTS
                      WHERE CONSTRAINT_TYPE IN
                          ( 'UNIQUE', 'PRIMARY KEY', 'FOREIGN KEY' ) ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the constraint being described.

2) The values of TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME are the catalog name, unqualified schema name, qualified identifier of the table name, and the column name of the column that participates in the unique, primary key, or foreign key constraint being described.

3) The value of ORDINAL_POSITION is the ordinal position of the specific column in the constraint being described. If the constraint described is a key of cardinality 1, then the value of ORDINAL_POSITION is always 1. If the constraint being described is a foreign key constraint, then ORDINAL_POSITION also identifies the position within the uniqueness constraint of the column that this column references.

23.3.17 REFERENTIAL_CONSTRAINTS base table

Function

The REFERENTIAL_CONSTRAINTS table has one row for each row in the TABLE_CONSTRAINTS table that has a CONSTRAINT_TYPE value of ''FOREIGN KEY''.

Definition

```
CREATE TABLE REFERENTIAL_CONSTRAINTS
       (
       CONSTRAINT_CATALOG           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME              INFORMATION_SCHEMA.SQL_IDENTIFIER,

       UNIQUE_CONSTRAINT_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT UNIQUE_CONSTRAINT_CATALOG_NOT_NULL NOT NULL,
       UNIQUE_CONSTRAINT_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT UNIQUE_CONSTRAINT_SCHEMA_NOT_NULL NOT NULL,
       UNIQUE_CONSTRAINT_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT UNIQUE_CONSTRAINT_NAME_NOT_NULL NOT NULL,
       MATCH_OPTION                 INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT REFERENTIAL_MATCH_OPTION_NOT_NULL NOT NULL
         CONSTRAINT REFERENTIAL_MATCH_OPTION_CHECK
           CHECK ( MATCH_OPTION IN ( 'NONE', 'PARTIAL', 'FULL' ) ),
       UPDATE_RULE                  INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT REFERENTIAL_UPDATE_RULE_NOT_NULL NOT NULL
         CONSTRAINT REFERENTIAL_UPDATE_RULE_CHECK
           CHECK ( UPDATE_RULE IN
                   ('CASCADE', 'SET NULL', 'SET DEFAULT', 'RESTRICT', 'NO ACTION' ) ),
       DELETE_RULE                  INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT REFERENTIAL_DELETE_RULE_NOT_NULL NOT NULL
         CONSTRAINT REFERENTIAL_DELETE_RULE_CHECK
           CHECK ( DELETE_RULE IN
                   ('CASCADE', 'SET NULL', 'SET DEFAULT', 'RESTRICT', 'NO ACTION' ) ),

       CONSTRAINT REFERENTIAL_CONSTRAINTS_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),

       CONSTRAINT REFERENTIAL_CONSTRAINTS_CONSTRAINT_TYPE_CHECK
         CHECK ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
               IN ( SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                      FROM TABLE_CONSTRAINTS
                      WHERE CONSTRAINT_TYPE = 'FOREIGN KEY' ) ),

       CONSTRAINT UNIQUE_CONSTRAINT_CHECK_REFERENCES_UNIQUE_CONSTRAINT
         CHECK ( UNIQUE_CONSTRAINT_CATALOG <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
               OR
               ( ( UNIQUE_CONSTRAINT_CATALOG, UNIQUE_CONSTRAINT_SCHEMA,
                   UNIQUE_CONSTRAINT_NAME ) IN
                     ( SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                       FROM TABLE_CONSTRAINTS
                       WHERE CONSTRAINT_TYPE
                               IN ( 'UNIQUE', 'PRIMARY KEY' ) ) ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the constraint being described.

2) The values of UNIQUE_CONSTRAINT_CATALOG, UNIQUE_CONSTRAINT_SCHEMA, and UNIQUE_CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the unique or primary key constraint applied to the referenced column list being described.

3) The values of MATCH_OPTION have the following meanings:

NONE	No <match type> was specified.
PARTIAL	A <match type> of PARTIAL was specified.
FULL	A <match type> of FULL was specified.

4) The values of UPDATE_RULE have the following meanings for a referential constraint that has an <update rule>:

NO ACTION	A <referential action> of NO ACTION was specified.
RESTRICT	A <referential action> of RESTRICT was specified.
CASCADE	A <referential action> of CASCADE was specified.
SET NULL	A <referential action> of SET NULL was specified.
SET DEFAULT	A <referential action> of SET DEFAULT was specified.

5) The values of DELETE_RULE have the following meanings for a referential constraint that has a <delete rule>:

NO ACTION	A <referential action> of NO ACTION was specified.
RESTRICT	A <referential action> of RESTRICT was specified.
CASCADE	A <referential action> of CASCADE was specified.
SET NULL	A <referential action> of SET NULL was specified.
SET DEFAULT	A <referential action> of SET DEFAULT was specified.

23.3.18 CHECK_CONSTRAINTS base table

Function

The CHECK_CONSTRAINTS table has one row for each domain constraint, table check constraint, and assertion.

Definition

```
CREATE TABLE CHECK_CONSTRAINTS
       (
       CONSTRAINT_CATALOG INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA  INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CHECK_CLAUSE       INFORMATION_SCHEMA.CHARACTER_DATA,

       CONSTRAINT CHECK_CONSTRAINTS_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),

       CONSTRAINT CHECK_CONSTRAINTS_SOURCE_CHECK
         CHECK ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
                     IN
                       ( SELECT * FROM (
                           SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME

                           FROM ASSERTIONS
                           UNION
                           SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME

                           FROM TABLE_CONSTRAINTS
                           UNION
                           SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME

                           FROM DOMAIN_CONSTRAINTS ) ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the constraint being described.

2) Case:

   a) If the character representation of the <search condition> contained in the <check constraint definition>, <domain constraint definition>, or <assertion definition> that defined the check constraint being described can be represented without truncation, then the value of CHECK_CLAUSE is that character representation.

   b) Otherwise, the value of CHECK_CLAUSE is the null value.

   Note: Any implicit <column reference>s that were contained in the <search condition> associated with a <check constraint definition> or an <assertion definition> are replaced by explicit <column reference>s in CHECK_CONSTRAINTS.

23.3.19 CHECK_TABLE_USAGE base table

Function

The CHECK_TABLE_USAGE table has one row for each table referenced by the <search condition> of a check constraint, domain constraint, or assertion.

Definition

```
CREATE TABLE CHECK_TABLE_USAGE
       (
       CONSTRAINT_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT CHECK_TABLE_USAGE_PRIMARY_KEY
       PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
                     TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ),

       CONSTRAINT CHECK_TABLE_USAGE_FOREIGN_KEY_CHECK_CONSTRAINTS
       FOREIGN KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
         REFERENCES CHECK_CONSTRAINTS,

       CONSTRAINT CHECK_TABLE_USAGE_CHECK_REFERENCES_TABLES
           CHECK ( TABLE_CATALOG
                     <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
              OR
                  ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
                  ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                    FROM TABLES ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the constraint being described.

2) The values of TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of a table that is referenced by the constraint being described.

23.3.20 CHECK_COLUMN_USAGE base table

Function

The CHECK_COLUMN_USAGE table has one row for each column referenced by the <search condition> of a check constraint, domain constraint, or assertion.

Definition

```
CREATE TABLE CHECK_COLUMN_USAGE
       (
       CONSTRAINT_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT CHECK_COLUMN_USAGE_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
                       TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ),

       CONSTRAINT CHECK_COLUMN_USAGE_FOREIGN_KEY_CHECK_CONSTRAINTS
         FOREIGN KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
           REFERENCES CHECK_CONSTRAINTS,

       CONSTRAINT CHECK_COLUMN_USAGE_CHECK_REFERENCES_COLUMNS
         CHECK ( TABLE_CATALOG
                   <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
                 OR
                 ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ) IN
                   ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
                     FROM COLUMNS ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the constraint being described.

2) The values of TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of a column that is referenced by the constraint being described.

23.3.21 ABSTRACT_DATA_TYPE_PRIVILEGES base table

Function

The ABSTRACT_DATA_TYPE_PRIVILEGES table has one row for each abstract data type privilege descriptor. It effectively contains a representation of the privilege descriptors.

Definition

```
CREATE TABLE ABSTRACT_DATA_TYPE_PRIVILEGES
       (
         GRANTOR                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
         GRANTEE                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_LANGUAGE   INFORMATION_SCHEMA.SQL_IDENTIFIER,
         PRIVILEGE_TYPE                INFORMATION_SCHEMA.CHARACTER_DATA
         CHECK ( PRIVILEGE_TYPE = 'DATA TYPE USAGE' ),
         IS_GRANTABLE                  INFORMATION_SCHEMA.CHARACTER_DATA NOT NULL
           CHECK ( IS_GRANTABLE IN ( 'YES', 'NO' ) ),
         PRIMARY KEY(GRANTOR, GRANTEE,
                     ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                     ABSTRACT_DATA_TYPE_NAME, ABSTRACT_DATA_TYPE_LANGUAGE,
                     PRIVILEGE_TYPE ),
         FOREIGN KEY ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME )
           REFERENCES ABSTRACT_DATA_TYPES,
         FOREIGN KEY ( GRANTOR )
           REFERENCES USERS,
         FOREIGN KEY ( GRANTEE )
           REFERENCES USERS
       )
```

Description

1) A row is inserted into this table when a <grant statement> is executed, unless the necessary row already exists, in which case the existing row may be modified to change the IS_GRANTABLE column. One or more rows are deleted from this table when a <revoke statement> is executed.

2) The value of GRANTOR is the <authorization identifier> of the user who granted access privileges on the DATA TYPE USAGE privilege being described to the user identified by the value of GRANTEE.

3) The value of GRANTEE is the <authorization identifier>, possibly ''PUBLIC'', of some <grantee> contained in a <grant statement>.

4) The value of PRIVILEGE_TYPE has the following meaning:

DATA TYPE USAGE	The user has DATA TYPE USAGE privilege on this abstract data type.

23.3.22 ASSERTIONS base table

Function

The ASSERTIONS table has one row for each assertion. It effectively contains a representation of the assertion descriptors.

Definition

```
CREATE TABLE ASSERTIONS
       (
       CONSTRAINT_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       IS_DEFERRABLE         INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT ASSERTIONS_IS_DEFERRABLE_NOT_NULL NOT<REFEREN
       INITIALLY_DEFERRED    INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT ASSERTIONS_INITIALLY_DEFERRED_NOT_NULL NOT<RE
       CHECK_TIME            INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT ASSERTIONS_CHECK_TIME_CHECK
           CHECK ( CHECK_TIME IN ('IMMEDIATE', 'DEFERRED' ) ),

       CONSTRAINT ASSERTIONS_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),

       CONSTRAINT ASSERTIONS_FOREIGN_KEY_CHECK_CONSTRAINTS
         FOREIGN KEY (CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
           REFERENCES CHECK_CONSTRAINTS,

       CONSTRAINT ASSERTIONS_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT ASSERTIONS_DEFERRED_CHECK
         CHECK ( ( IS_DEFERRABLE, INITIALLY_DEFERRED ) IN
            VALUES ( ( 'NO',  'NO' ),
                     ( 'YES', 'NO' ),
                     ( 'YES', 'YES' ) ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the assertion being described.

2) The values of IS_DEFERRABLE have the following meanings:

YES	The assertion is deferrable.
NO	The assertion is not deferrable.

3) The values of INITIALLY_DEFERRED have the following meanings:

YES	The assertion is initially deferred.
NO	The assertion is initially immediate.

23.3.23 PRIVILEGE_DESCRIPTORS base table

Function

Contains a representation of the privilege descriptors.

****Editor's Note****

Does this table still belong in the Definition Schema? SQL-92 doesn't have it, but has TABLE_PRIVILEGES, etc. instead, which this document also has.

Definition

```
CREATE TABLE PRIVILEGE_DESCRIPTORS
       (
       GRANTOR                          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       GRANTEE                          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_NAME                      INFORMATION_SCHEMA.SQL_IDENTIFIER,
       PRIVILEGE_TYPE                   INFORMATION_SCHEMA.CHARACTER_DATA,
       IS_GRANTABLE                     INFORMATION_SCHEMA.CHARACTER_DATA
           CHECK ( IS_GRANTABLE IN ('YES', 'NO' )

       PRIMARY KEY ( GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
                     GRANTOR, PRIVILEGE_TYPE, COLUMN_NAME ),

       FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
           REFERENCES TABLES,

       FOREIGN KEY ( GRANTOR )
           REFERENCES USERS,

       CHECK ( GRANTEE IN ( SELECT ROLE_NAME FROM ROLES )
            OR GRANTEE IN ( SELECT USER_NAME FROM USERS ) )

       CHECK ( ( ( COLUMN_NAME, TABLE_NAME ) IN
                     ( SELECT COLUMN_NAME, TABLE_NAME FROM COLUMNS )
                 AND PRIVILEGE_TYPE IN ( 'UPDATE', 'INSERT', 'DELETE' ) )
               OR
                 ( COLUMN_NAME = 'ALL'
                   AND PRIVILEGE_TYPE IN ( 'SELECT', 'INSERT', 'DELETE', 'UPDATE',
                                           'TRIGGER', 'REFERENCES' ) )
               )
       )
```

Description

1) Rows are inserted into this table when a <grant statement> is executed, unless the necessary row already exists, in which case the existing row may be modified to change the IS_GRANTABLE column. One or more rows are deleted from this table when a <revoke statement> is executed.

2) The value of GRANTOR is the <authorization identifier> of the user who granted access privileges on the table being described to the user or role identified by the value of GRANTEE.

3) The value of GRANTEE is an <authorization identifier>, possibly 'PUBLIC', or <role name> specified as a <grantee> contained in a <grant statement>.

4) The values of TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME give the qualified name of the table on which the privilege being described has been granted.

5) The value of COLUMN_NAME is the qualified name of the column of the table, if any, on which the privilege being described has been granted. COLUMN_NAME will contain 'ALL' if no column privilege on the table is specified.

6) If a column privilege is not specified then the values of PRIVILEGE_TYPE have the following meanings:

SELECT	The user has SELECT privileges on this table.
DELETE	The user has DELETE privileges on this table.
INSERT	The user has INSERT privilege on all columns and will automatically be granted INSERT privilege on any columns that may be added to this table in the future.
UPDATE	The user has UPDATE privilege on all columns and will automatically be granted UPDATE privileges on any columns that may be added to this table in the future.
REFERENCES	The user has REFERENCES privilege on all columns and will automatically be granted REFERENCES privileges on any columns that may be added to this table in the future.
TRIGGER	The user has TRIGGER privilege on this table.

7) If a column privilege is specified then the values of PRIVILEGE_TYPE have the following meanings:

SELECT	The user has SELECT privileges on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.
INSERT	The user has INSERT privileges on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.
UPDATE	The user has UPDATE privileges on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.
REFERENCES	The user has REFERENCES privileges on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.

8) The values of IS_GRANTABLE have the following meanings:

YES	The privilege being described was granted WITH GRANT OPTION and is thus grantable.
NO	The privilege being described was not granted WITH GRANT OPTION and is thus not grantable.

Note: Privileges granted to roles cannot be granted WITH GRANT OPTION and hence are never grantable.

23.3.24 ROLE_AUTHORIZATION_DESCRIPTORS base table

Function

Contains a representation of the role authorization descriptors.

Definition

```
CREATE TABLE ROLE_AUTHORIZATION_DESCRIPTORS
       (
       ROLE_NAME                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       GRANTEE                         INFORMATION_SCHEMA.SQL_IDENTIFIER
         CHECK ( GRANTEE IN ( SELECT ROLE_NAME FROM ROLES )
               OR GRANTEE IN ( SELECT USER_NAME FROM USERS ) ),
       IS_GRANTABLE                    INFORMATION_SCHEMA.CHARACTER_DATA
         CHECK ( IS_GRANTABLE IN ( 'YES', 'NO' ) ),

       PRIMARY KEY ( ROLE_NAME, GRANTEE, IS_GRANTABLE ),

       FOREIGN KEY ( ROLE_NAME )
         REFERENCES ROLES
       )
```

Description

1) A row is (or rows are) inserted into this table whenever a <grant role statement> or <role definition> is executed unless the necessary row already exists, in which case the existing row may be modified to change the IS_GRANTABLE column. A row is (or rows are) deleted from this table whenever a <revoke role statement> or <drop role> is executed.

2) The value of ROLE_NAME is the <role name> of some <role granted> by the <grant role statement> or the <role name> of a <role definition>.

3) The value of GRANTEE is an <authorization identifier>, possibly PUBLIC, or <role name> specified as a <grantee> contained in a <grant role statement>, or the <authorization identifier> of the current SQL-session when the <role definition> is executed.

4) The values of IS_GRANTABLE have the following meanings:

YES	The described role is grantable.
NO	The described role is not grantable.

   A role is grantable if the WITH ADMIN OPTION is specified in the <grant role statement> or a <role definition> is executed.

23.3.25 NULL_CLASSES base table

Function

The NULL_CLASSES table has one row for each null class.

Definition

```
CREATE TABLE NULL_CLASSES
       (
         NULL_CLASS_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         NULL_CLASS_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER,
         NULL_CLASS_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,
         CONSTRAINT NULL_CLASSES_PRIMARY_KEY
           PRIMARY KEY ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME ),
         CONSTRAINT NULL_CLASSES_FOREIGN_KEY_SCHEMATA
           FOREIGN KEY ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME )
             REFERENCES SCHEMATA
       )
```

Description

1) A row is inserted in this table whenever a <null class definition> is executed; a row is deleted whenever a <drop null class statement> is executed.

2) The value of NULL_CLASS_CATALOG and NULL_CLASS_SCHEMA is the name of the schema in which the null class is defined.

23.3.26 NULL_STATES base table

Function

The NULL_STATES table has one row for each null state.

Definition

```
CREATE TABLE NULL_STATES
       (
         NULL_CLASS_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         NULL_CLASS_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER,
         NULL_CLASS_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,
         NULL_STATE_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ORDINAL_POSITION       INFORMATION_SCHEMA.CARDINAL_NUMBER,
         CONSTRAINT NULL_STATES_PRIMARY_KEY
           PRIMARY KEY ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME,
                         NULL_STATE_NAME ),
         CONSTRAINT NULL_CLASSES_FOREIGN_KEY_NULL_CLASSES
           FOREIGN KEY ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME )
             REFERENCES NULL_CLASSES,
         CONSTRAINT NULL_CLASSES_UNIQUE
           UNIQUE ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA,
                    NULL_CLASS_NAME, ORDINAL_POSITION )
       )
```

Description

1) A row is inserted in this table for each null state in a null class whenever a <null class definition> is executed. Whenever a <drop null class statement> is executed, the rows in this table that correspond to the null states of that null class are deleted.

2) The value of NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, and NULL_CLASS_NAME is the name of the null class that contains the null state.

3) NULL_STATE_NAME is the name of the null state.

   Note: The fact that a null state in one null class may have the same name as some null state in some other null class is of no significance.

4) ORDINAL_POSITION is the position of the null state within the null class.

23.3.27 TABLE_PRIVILEGES base table

Function

The TABLE_PRIVILEGES table has one row for each table privilege descriptor. It effectively contains a representation of the table privilege descriptors.

Definition

```
CREATE TABLE TABLE_PRIVILEGES
       (
       GRANTOR                INFORMATION_SCHEMA.SQL_IDENTIFIER,
       GRANTEE                INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,
       PRIVILEGE_TYPE              INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_PRIVILEGE_TYPE_CHECK
           CHECK ( PRIVILEGE_TYPE IN
                     ( 'SELECT', 'INSERT', 'DELETE', 'UPDATE'
                       'TRIGGER', 'REFERENCES' ) ),
       IS_GRANTABLE           INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_PRIVILEGE_GRANTABLE_NOT_NULL NOT<REFERE
         CONSTRAINT TABLE_PRIVILEGE_GRANTABLE_CHECK
           CHECK ( IS_GRANTABLE IN ( 'YES', 'NO' ) ),

       CONSTRAINT TABLE_PRIVILEGE_PRIMARY_KEY
         PRIMARY KEY ( GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
                       PRIVILEGE_TYPE ),

       CONSTRAINT TABLE_PRIVILEGE_FOREIGN_KEY_TABLES
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
           REFERENCES TABLES,
       CONSTRAINT TABLE_PRIVILEGE_GRANTOR_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTOR )
           REFERENCES USERS,


       CONSTRAINT TABLE_PRIVILEGE_GRANTEE_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTEE )
           REFERENCES USERS
       )
```

Description

1) The value of GRANTOR is the <authorization identifier> of the user who granted table privileges, on the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME, to the user identified by the value of GRANTEE for the table privilege being described.

2) The value of GRANTEE is the <authorization identifier> of some user, or ''PUBLIC'' to indicate all users, to whom the table privilege being described is granted.

3) The values of TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the table on which the privilege being described has been granted.

4) The values of PRIVILEGE_TYPE have the following meanings:

SELECT	The user has SELECT privileges on the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME.
DELETE	The user has DELETE privileges on the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME.
INSERT	The user will automatically be granted INSERT privileges on any columns that may be added to the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME in the future.
UPDATE	The user will automatically be granted UPDATE privileges on any columns that may be added to the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME in the future.
REFERENCES	The user will automatically be granted REFERENCES privileges on any columns that may be added to the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME in the future.
TRIGGER	The user has TRIGGER privilege on the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME.

5) The values of IS_GRANTABLE have the following meanings:

YES	The privilege being described was granted WITH GRANT OPTION and is thus grantable.
NO	The privilege being described was not granted WITH GRANT OPTION and is thus not grantable.

23.3.28 COLUMN_PRIVILEGES base table

Function

The COLUMN_PRIVILEGES table has one row for each column privilege descriptor. It effectively contains a representation of the column privilege descriptors.

Definition

```
CREATE TABLE COLUMN_PRIVILEGES
       (
       GRANTOR          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       GRANTEE          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_NAME      INFORMATION_SCHEMA.SQL_IDENTIFIER,
       PRIVILEGE_TYPE   INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT COLUMN_PRIVILEGE_TYPE_CHECK
           CHECK ( PRIVILEGE_TYPE IN ( 'SELECT', 'INSERT', 'UPDATE', 'REFERENCES' ) ),
       IS_GRANTABLE   INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT COLUMN_PRIVILEGE_IS_GRANTABLE_NOT_NULL NOT NULL
         CONSTRAINT COLUMN_PRIVILEGE_IS_GRANTABALE_CHECK
           CHECK ( IS_GRANTABLE IN ( 'YES', 'NO' ) ),

       CONSTRAINT COLUMN_PRIVILEGE_PRIMARY_KEY
         PRIMARY KEY
           ( GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
             PRIVILEGE_TYPE, COLUMN_NAME ),

       CONSTRAINT COLUMN_PRIVILEGE_FOREIGN_KEY_COLUMNS
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME )
           REFERENCES COLUMNS,

       CONSTRAINT COLUMN_PRIVILEGE_GRANTOR_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTOR )
           REFERENCES USERS,

       CONSTRAINT COLUMN_PRIVILEGE_GRANTEE_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTEE )
           REFERENCES USERS
       )
```

Description

1) The value of GRANTOR is the <authorization identifier> of the user who granted column privileges, on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME, to the user identified by the value of GRANTEE for the column privilege being described.

2) The value of GRANTEE is the <authorization identifier> of some user, or ''PUBLIC'' to indicate all users, to whom the column privilege being described is granted.

3) The values of TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the column to which the privilege being described was granted.

4) The values of PRIVILEGE_TYPE have the following meanings:

SELECT	The user has SELECT privilege on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.
INSERT	The user has INSERT privilege on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.
UPDATE	The user has UPDATE privilege on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.
REFERENCE	The user has REFERENCES privilege on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.

5) The values of IS_GRANTABLE have the following meanings:

YES	The privilege being described was granted WITH GRANT OPTION and is thus grantable.
NO	The privilege being described was not granted WITH GRANT OPTION and is thus not grantable.

23.3.29 USAGE_PRIVILEGES base table

Function

The USAGE_PRIVILEGES table has one row for each usage privilege descriptor. It effectively contains a representation of the usave privilege descriptors.

Definition

```
CREATE TABLE USAGE_PRIVILEGES
       (
       GRANTOR        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       GRANTEE        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       OBJECT_CATALOG INFORMATION_SCHEMA.SQL_IDENTIFIER,
       OBJECT_SCHEMA  INFORMATION_SCHEMA.SQL_IDENTIFIER,
       OBJECT_NAME    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       OBJECT_TYPE    INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT USAGE_PRIVILEGES_OBJECT_TYPE_CHECK
           CHECK ( OBJECT_TYPE IN
                 ( 'DOMAIN', 'CHARACTER SET', 'COLLATION', 'TRANSLATION' ) ),
       IS_GRANTABLE   INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT USAGE_PRIVILEGES_IS_GRANTABLE_NOT_NULL NOT NULL
         CONSTRAINT USAGE_PRIVILEGES_IS_GRANTABLE_CHECK
           CHECK ( IS_GRANTABLE IN ( 'YES', 'NO' ) ),

       CONSTRAINT USAGE_PRIVILEGES_PRIMARY_KEY
         PRIMARY KEY ( GRANTOR, GRANTEE, OBJECT_CATALOG, OBJECT_SCHEMA,
                       OBJECT_NAME, OBJECT_TYPE ),

       CONSTRAINT USAGE_PRIVILEGES_CHECK_REFERENCES_OBJECT
         CHECK ( ( OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE ) IN
         ( SELECT DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,'DOMAIN'
             FROM DOMAINS
           UNION
           SELECT CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
                      'CHARACTER SET'
             FROM CHARACTER_SETS
           UNION
           SELECT COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, 'COLLATION'
             FROM COLLATIONS
           UNION
           SELECT TRANSLATION_CATALOG, TRANSLATION_SCHEMA, TRANSLATION_NAME,
                      'TRANSLATION'
             FROM TRANSLATIONS
           UNION
           SELECT NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME, 'NULL CLASS'
             FROM NULL_CLASSES ) ),

       CONSTRAINT USAGE_PRIVILEGES_GRANTOR_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTOR )
           REFERENCES USERS,

       CONSTRAINT USAGE_PRIVILEGES_GRANTEE_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTEE )
           REFERENCES USERS
       )
```

Description

1) The value of GRANTOR is the <authorization identifier> of the user who granted usage privileges, on the object of the type identified by OBJECT_TYPE that is identified by OBJECT_CATALOG, OBJECT_SCHEMA, and OBJECT_NAME, to the user identified by the value of GRANTEE for the usage privilege being described.

2) The value of GRANTEE is the <authorization identifier> of some user, or ''PUBLIC'' to indicate all users, to whom the usage privilege being described is granted.

3) The values of OBJECT_CATALOG, OBJECT_SCHEMA, and OBJECT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the object to which the privilege applies.

4) The values of OBJECT_TYPE has the following meanings:

DOMAIN	The object to which the privilege applies is a domain.
CHARACTER SET	The object to which the privilege applies is a character set.
COLLATION	The object to which the privilege applies is a collation.
TRANSLATION	The object to which the privilege applies is a translation.
NULL CLASS	The object to which the privilege applies is a null class.

5) The values of IS_GRANTABLE have the following meanings:

YES	The privilege being described was granted WITH GRANT OPTION and is thus grantable.
NO	The privilege being described was not granted WITH GRANT OPTION and is thus not grantable.

23.3.30 CHARACTER_SETS base table

Function

The CHARACTER_SETS table has one row for each character set descriptor.

Definition

```
CREATE TABLE CHARACTER_SETS
       (
       CHARACTER_SET_CATALOG         INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CHARACTER_SET_SCHEMA          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CHARACTER_SET_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       FORM_OF_USE                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
       NUMBER_OF_CHARACTERS          INFORMATION_SCHEMA.CARDINAL_NUMBER,
       DEFAULT_COLLATE_CATALOG       INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT CHARACTER_SETS_DEFAULT_COLLATE_CATALOG_NOT_NULL NOT NULL,
       DEFAULT_COLLATE_SCHEMA        INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT CHARACTER_SETS_DEFAULT_COLLATE_SCHEMA_NOT_NULL NOT NULL,
       DEFAULT_COLLATE_NAME          INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT CHARACTER_SETS_DEFAULT_COLLATE_NAME_NOT_NULL NOT NULL,

       CONSTRAINT CHARACTER_SETS_PRIMARY_KEY
         PRIMARY KEY ( CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME ),


       CONSTRAINT CHARACTER_SETS_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT CHARACTER_SETS_CHECK_REFERENCES_COLLATIONS
         CHECK ( CHARACTER_SET_CATALOG
                 <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
               OR
                 ( DEFAULT_COLLATE_CATALOG, DEFAULT_COLLATE_SCHEMA,
                   DEFAULT_COLLATE_NAME ) IN
                     ( SELECT COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME
                       FROM COLLATIONS ) )
       )
```

Description

1) The values of CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the character set being described.

2) The value of FORM_OF_USE is the name of the form-of-use of the character set.

3) The value of NUMBER_OF_CHARACTERS is the number of characters in the character repertoire.

4) Case:

   a) If the default collation for the character repertoire is the order of characters in the repertoire, then the values of DEFAULT_COLLATE_CATALOG and DEFAULT_COLLATE_SCHEMA are the values of CHARACTER_SET_CATALOG and CHARACTER_SET_SCHEMA, respectively, and the value of DEFAULT_COLLATE_NAME is implementation-dependent.

b) Otherwise, the values of DEFAULT_COLLATE_CATALOG, DEFAULT_COLLATE_SCHEMA, and DEFAULT_COLLATE_NAME are catalog name, unqualified schema name, and qualified identifier, respectively, of the default collation.

5) There is a row in this table for the character set INFORMATION_SCHEMA.SQL_TEXT. In that row:

a) CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are the name of the catalog, 'INFORMATION_SCHEMA', and 'SQL_TEXT', respectively.

b) FORM_OF_USE is implementation-defined.

c) NUMBER_OF_CHARACTERS is implementation-defined.

d) DEFAULT_COLLATE_CATALOG, DEFAULT_COLLATE_SCHEMA, and DEFAULT_COLLATE_NAME are the name of the catalog, 'INFORMATION_SCHEMA', and 'SQL_TEXT', respectively.

23.3.31 COLLATIONS base table

Function

The COLLATIONS table has one row for each character collation descriptor.

Definition

```
CREATE TABLE COLLATIONS
       (
       COLLATION_CATALOG INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLLATION_SCHEMA  INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLLATION_NAME    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CHARACTER_SET_CATALOG   INFORMATION_SCHEMA.SQL_IDENTIFIER COMMENT>(18)
         CONSTRAINT COLLATIONS_CHARACTER_SET_CATALOG_NOT_NULL NOT NULL,
       CHARACTER_SET_SCHEMA    INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT COLLATIONS_CHARACTER_SET_SCHEMA_NOT_NULL NOT NULL,
       CHARACTER_SET_NAME      INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT COLLATIONS_CHARACTER_SET_NAME_NOT_NULL NOT NULL,
       PAD_ATTRIBUTE           INFORMATION_SCHEMA.CHARACTER_DATA
           CONSTRAINT COLLATIONS_PAD_ATTRIBUTE_CHECK
             CHECK ( PAD_ATTRIBUTE IN ( 'NO PAD', 'PAD SPACE' ) ),
       COLLATION_TYPE    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLLATION_DEFINITION    INFORMATION_SCHEMA.CHARACTER_DATA,
       COLLATION_DICTIONARY    INFORMATION_SCHEMA.CHARACTER_DATA,

       CONSTRAINT COLLATIONS_CHECK
         CHECK ( ( COLLATION_TYPE = 'DICTIONARY' AND
                   ( COLLATION_DEFINITION IS NULL AND COLLATION_DICTIONARY IS NOT NULL ) )

               OR
                    ( COLLATION_TYPE = 'SEQUENCE' AND
                      ( COLLATION_DICTIONARY IS NULL AND COLLATION_DEFINITION IS NOT NULL ) )

                )

       CONSTRAINT COLLATIONS_PAD_PRIMARY_KEY
         PRIMARY KEY ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ),

       CONSTRAINT COLLATIONS_PAD_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( COLLATION_CATALOG, COLLATION_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT COLLATIONS_CHECK_REFERENCES_CHARACTER_SETS
         CHECK ( COLLATION_CATALOG
                   <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
                 OR
                   ( CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME )
                      IN
                        ( SELECT CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA,
                                 CHARACTER_SET_NAME
                          FROM CHARACTER_SETS ) )
       )
```

Description

1) The values of COLLATION_CATALOG, COLLATION_SCHEMA, and COLLATION_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the collation being described.

2) The values of CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the character set on which the collation is defined.

3) The values of PAD_ATTRIBUTE have the following meanings:

NO PAD	The collation being described has the NO PAD attribute.
PAD SPACE	The collation being described has the PAD SPACE attribute.

4) The values of COLLATION_TYPE have the following meanings:

DICTIONARY	The collation being described is a dictionary collation.
SEQUENCE	The collation being described is a collating sequence.

5) Case:

   a) If the value of COLLATION_TYPE is 'SEQUENCE', then the value of COLLATION_DEFINITION is a character representation of an <internal collation source> for the collation, as though the collation had been defined with an <internal collation source>.

   b) If the value of COLLATION_TYPE is 'DICTIONARY', then the value of COLLATION_DICTIONARY is an implementation-defined name of a collation dictionary.

6) There is a row in this table for the collation INFORMATION_SCHEMA.SQL_TEXT. That row contains the definition of the collation corresponding to the default collation for the characters in the character set SQL_TEXT. In that row:

   a) COLLATION_CATALOG, COLLATION_SCHEMA, and COLLATION_NAME are the name of the catalog, 'INFORMATION_SCHEMA', and 'SQL_TEXT', respectively.

   b) CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are the name of the catalog, 'INFORMATION_SCHEMA', and 'SQL_TEXT', respectively.

   c) PAD_ATTRIBUTE is implementation-defined.

   d) COLLATION_TYPE is 'SEQUENCE'.

   e) COLLATION_DICTIONARY is null.

   f) COLLATION_DEFINITION is implementation-defined.

23.3.32 TRANSLATIONS base table

Function

The TRANSLATIONS table has one row for each character translation descriptor.

Definition

```
CREATE TABLE TRANSLATIONS
       (
       TRANSLATION_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TRANSLATION_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TRANSLATION_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       SOURCE_CHARACTER_SET_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TRANSLATIONS_SOURCE_CHARACTER_SET_CATALOG_NOT_NULL NOT NULL,
       SOURCE_CHARACTER_SET_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TRANSLATIONS_SOURCE_CHARACTER_SET_SCHEMA_NOT_NULL NOT NULL,
       SOURCE_CHARACTER_SET_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TRANSLATIONS_SOURCE_CHARACTER_SET_NAME_NOT_NULL NOT NULL,
       TARGET_CHARACTER_SET_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TRANSLATIONS_TARGET_CHARACTER_SET_CATALOG_NOT_NULL NOT NULL,
       TARGET_CHARACTER_SET_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TRANSLATIONS_TARGET_CHARACTER_SET_SCHEMA_NOT_NULL NOT NULL,
       TARGET_CHARACTER_SET_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TRANSLATIONS_TARGET_CHARACTER_SET_NAME_NOT_NULL NOT NULL,

       CONSTRAINT TRANSLATIONS_PRIMARY_KEY
         PRIMARY KEY ( TRANSLATION_CATALOG, TRANSLATION_SCHEMA, TRANSLATION_NAME ),

       CONSTRAINT TRANSLATIONS_FOREIGN_KEYSCHEMATA
         FOREIGN KEY ( TRANSLATION_CATALOG, TRANSLATION_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT TRANSLATIONS_CHECK_REFERENCES_SOURCE
         CHECK ( SOURCE_CHARACTER_SET_CATALOG
                 <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
               OR
                 ( SOURCE_CHARACTER_SET_CATALOG, SOURCE_CHARACTER_SET_SCHEMA,
                   SOURCE_CHARACTER_SET_NAME ) IN
                     ( SELECT CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA,
                              CHARACTER_SET_NAME
                       FROM CHARACTER_SETS ) ),

       CONSTRAINT TRANSLATIONS_CHECK_REFERENCES_TARGET
         CHECK ( TARGET_CHARACTER_SET_CATALOG
                 <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
               OR
                 ( TARGET_CHARACTER_SET_CATALOG, TARGET_CHARACTER_SET_SCHEMA,
                     TARGET_CHARACTER_SET_NAME ) IN
                       ( SELECT CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA,
                                CHARACTER_SET_NAME
                         FROM CHARACTER_SETS ) )
       )
```

Description

1) The values of TRANSLATION_CATALOG, TRANSLATION_SCHEMA, and TRANSLATION_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the translation being described.

2) The values of SOURCE_CHARACTER_SET_CATALOG, SOURCE_CHARACTER_SET_SCHEMA, and SOURCE_CHARACTER_SET_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the character set specified as the source for the translation.

3) The values of TARGET_CHARACTER_SET_CATALOG, TARGET_CHARACTER_SET_SCHEMA, and TARGET_CHARACTER_SET_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the character set specified as the target for the translation.

4) The value of TRANSLATION_DEFINITION is a character representation of an <internal translation source> for the translation, as though the translation had been defined with an <internal translation source>.

ANSI Only–SQL3

23.3.33 ROUTINES base table

Function

The ROUTINES base table has one row for each To Be Supplied.

Definition

```
CREATE TABLE ROUTINES
       (
         ROUTINE_CATALOG                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ROUTINE_SCHEMA                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ROUTINE_NAME                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       To Be Supplied

         PRIMARY KEY ( ROUTINE_CATALOG, ROUTINE_SCHEMA, ROUTINE_NAME ),
         FOREIGN KEY ( ROUTINE_CATALOG, ROUTINE_SCHEMA )
           REFERENCES SCHEMATA
       )
```

Description

1) The values of ROUTINE_CATALOG, ROUTINE_SCHEMA, and ROUTINE_NAME are the catalog name, schema name, and routine name of the routine being described.

2) To Be Supplied.

23.3.34 TRIGGERS base table

Function

The TRIGGERS base table has one row for each trigger. It effectively contains a representation of the trigger descriptors.

Definition

```
CREATE TABLE TRIGGERS
       (
         TRIGGER_CATALOG                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TRIGGER_SCHEMA                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TRIGGER_NAME                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
         EVENT_MANIPULATION                 INFORMATION_SCHEMA.CHARACTER_DATA
             CHECK ( EVENT_MANIPULATION IN
                         ( 'INSERT', 'DELETE', 'UPDATE' ) ),
         EVENT_OBJECT_CATALOG               INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,

         EVENT_OBJECT_SCHEMA                INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,

         EVENT_OBJECT_TABLE                 INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,

         CONDITION_TIMING                   INFORMATION_SCHEMA.CHARACTER_DATA
             CHECK ( CONDITION_TIMING IN
                         ( 'BEFORE', 'AFTER' ) ),
         CONDITION_REFERENCE_OLD_TABLE      INFORMATION_SCHEMA.SQL_IDENTIFIER,
         CONDITION_REFERENCE_NEW_TABLE      INFORMATION_SCHEMA.SQL_IDENTIFIER,
```

ANSI Only--SQL3

```
         COLUMN_LIST_IS_IMPLICIT            INFORMATION_SCHEMA.SQL_IDENTIFIER
           CONSTRAINT IS_IMPLICIT_CHECK
             CHECK ( COLUMN_LIST_IS_IMPLICIT IN ( 'YES', 'NO' )
                OR
                   COLUMN_LIST_IS_IMPLICIT IS NULL ),
```

```
         PRIMARY KEY ( TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME ),
         FOREIGN KEY ( TRIGGER_CATALOG, TRIGGER_SCHEMA )
```

ANSI Only--SQL3

```
           REFERENCES SCHEMATA,
         CONSTRAINT EVENT_MANIPULATION_UPDATE_CHECK
           CHECK ( ( EVENT_MANIPULATION <> 'UPDATE' AND
                     COLUMN_LIST_IS_IMPLICIT IS NULL )
                 OR
           CHECK ( ( EVENT_MANIPULATION = 'UPDATE' AND
                     COLUMN_LIST_IS_IMPLICIT IS NOT NULL )
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
           REFERENCES SCHEMATA
```

```
       )
```

Description

1) The values of TRIGGER_CATALOG, TRIGGER_SCHEMA, and TRIGGER_NAME are the catalog name, schema name, and trigger name of the trigger being described.

2) The values of EVENT_MANIPULATION have the following meaning:

INSERT	The <trigger event> is INSERT.
DELETE	The <trigger event> is DELETE.
UPDATE	The <trigger event> is UPDATE.

3) The values of EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA, and EVENT_OBJECT_TABLE are the qualified name of the <table name> of the trigger being described.

4) The values of CONDITION_TIMING have the following meaning:

BEFORE	The <trigger action time> is BEFORE.
AFTER	The <trigger action time> is AFTER.

5) The value of CONDITION_REFERENCE_OLD_TABLE is the <old value correlation name> of the trigger being described.

6) The value of CONDITION_REFERENCE_NEW_TABLE is the <new value correlation name> of the trigger being described.

ANSI Only–SQL3

7) The values of COLUMN_LIST_IS_IMPLICIT have the following meaning:

YES	The <trigger event> is UPDATE and the <trigger column list> is implicit.
NO	The <trigger event> is UPDATE and the <trigger column list> is explicit.
null	The <trigger event> is INSERT or DELETE.

23.3.35 TRIGGERED_ACTIONS base table

Function

The TRIGGERED_ACTIONS base table has one row for each triggered action in the catalog.

Definition

```
CREATE TABLE TRIGGERED_ACTIONS
       (
         TRIGGER_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TRIGGER_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TRIGGER_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,
```

ANSI Only--SQL3

```
         ACTION_ORDER             INFORMATION_SCHEMA.CARDINAL_NUMBER NOT NULL,
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
         ACTION_ORDER             INTEGER,
```

```
         ACTION_CONDITION         INFORMATION_SCHEMA.CHARACTER_DATA,
```

ANSI Only--SQL3

```
         ACTION_STATEMENT_LIST    INFORMATION_SCHEMA.CHARACTER_DATA NOT NULL,
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
         ACTION_STATEMENT_LIST    INFORMATION_SCHEMA.CHARACTER_DATA,

         ACTION_ORIENTATION       INFORMATION_SCHEMA.CHARACTER_DATA
             CHECK ( ACTION_ORIENTATION IN
                       ( 'ROW', 'STATEMENT' ) ),

         PRIMARY KEY ( TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME, ACTION_ORDER ),
         FOREIGN KEY ( TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME )
           REFERENCES TRIGGERS
       )
```

Description

1) The values of TRIGGER_CATALOG, TRIGGER_SCHEMA, and TRIGGER_NAME are the catalog name, schema name, and trigger name of the trigger being described.

ANSI Only–SQL3

2) The value of ACTION_ORDER is the ordinal position of the <triggered action> in the list of <triggered action>s of the trigger being described.

ISO Only—caused by ANSI changes not yet considered by ISO

3) The value of ACTION_ORDER is a description of each <triggered action> in the trigger being described.

4) The value of ACTION_CONDITION is a character representation of the <search condition> in the <triggered action> of the trigger being described.

5) ACTION_STATEMENT_LIST is a character representation of the <triggered SQL statement list> in the <triggered action> of the trigger being described.

6) The values of ACTION_ORIENTATION have the following meanings:

ROW	The <trigger action> specifies FOR EACH ROW.
STATEMENT	The <trigger action> specified FOR EACH STATEMENT.

23.3.36 TRIGGERED_COLUMNS base table

Function

The TRIGGERED_COLUMNS base table has one row for each column referenced by a trigger.

Definition

```
CREATE TABLE TRIGGERED_COLUMNS
       (
         TRIGGER_CATALOG            INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TRIGGER_SCHEMA             INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TRIGGER_NAME               INFORMATION_SCHEMA.SQL_IDENTIFIER,
         EVENT_OBJECT_CATALOG       INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
         EVENT_OBJECT_SCHEMA        INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
         EVENT_OBJECT_TABLE         INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
         EVENT_OBJECT_COLUMN        INFORMATION_SCHEMA.SQL_IDENTIFIER,

         PRIMARY KEY
           ( TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME, EVENT_OBJECT_COLUMN ),
         FOREIGN KEY ( TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME )
           REFERENCES TRIGGERS
         FOREIGN KEY
           ( EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA,
             EVENT_OBJECT_TABLE, EVENT_OBJECT_COLUMN )
           REFERENCES COLUMNS
       )
```

Description

1) The values of TRIGGER_CATALOG, TRIGGER_SCHEMA, and TRIGGER_NAME are the catalog name, schema name, and trigger name of the trigger being described.

2) The values of EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA, and EVENT_OBJECT_TABLE are the catalog name, schema name, and table name of the table containing the column being described.

23.3.37 SQL_LANGUAGES base table

Function

The SQL_LANGUAGES table has one row for each ISO and implementation-defined SQL language binding and programming language for which conformance is claimed.

Definition

```
CREATE TABLE SQL_LANGUAGES
       (
         SQL_LANGUAGE_SOURCE               INFORMATION_SCHEMA.CHARACTER_DATA
           CONSTRAINT SQL_LANGUAGES_SOURCE_NOT_NULL NOT NULL,
         SQL_LANGUAGE_YEAR                 INFORMATION_SCHEMA.CHARACTER_DATA,
         SQL_LANGUAGE_CONFORMANCE          INFORMATION_SCHEMA.CHARACTER_DATA,
         SQL_LANGUAGE_INTEGRITY            INFORMATION_SCHEMA.CHARACTER_DATA,
         SQL_LANGUAGE_IMPLEMENTATION       INFORMATION_SCHEMA.CHARACTER_DATA,
         SQL_LANGUAGE_BINDING_STYLE        INFORMATION_SCHEMA.CHARACTER_DATA,
         SQL_LANGUAGE_PROGRAMMING_LANGUAGE INFORMATION_SCHEMA.CHARACTER_DATA,

       CONSTRAINT SQL_LANGUAGES_STANDARD_VALID_CHECK
         CHECK ( ( SQL_LANGUAGE_SOURCE = 'ISO 9075' AND
                   SQL_LANGUAGE_YEAR IS NOT NULL AND
                   SQL_LANGUAGE_CONFORMANCE IS NOT NULL AND
                   SQL_LANGUAGE_IMPLEMENTATION IS NULL AND
                   ( ( SQL_LANGUAGE_YEAR = '1987' AND
                       SQL_LANGUAGE_CONFORMANCE IN ( '1', '2' ) AND
                       SQL_LANGUAGE_INTEGRITY IS NULL AND
                       ( ( SQL_LANGUAGE_BINDING_STYLE = 'DIRECT' AND
                           SQL_LANGUAGE_PROGRAMMING_LANGUAGE IS NULL )
                         OR
                         ( SQL_LANGUAGE_BINDING_STYLE IN ( 'EMBEDDED', 'MODULE' )
                           AND
                           SQL_LANGUAGE_PROGRAMMING_LANGUAGE IN
                             ( 'COBOL', 'FORTRAN', 'PASCAL', 'PLI' ) ) ) )
                     OR
                     ( SQL_LANGUAGE_YEAR = '1989' AND
                       SQL_LANGUAGE_CONFORMANCE IN ( '1', '2' ) AND
                       SQL_LANGUAGE_INTEGRITY IN ( 'NO', 'YES' ) AND
                       ( ( SQL_LANGUAGE_BINDING_STYLE = 'DIRECT' AND
                           SQL_LANGUAGE_PROGRAMMING_LANGUAGE IS NULL )
                         OR
                         ( SQL_LANGUAGE_BINDING_STYLE IN ( 'EMBEDDED', 'MODULE' )
                           AND
                           SQL_LANGUAGE_PROGRAMMING_LANGUAGE IN
                             ( 'COBOL', 'FORTRAN', 'PASCAL', 'PLI' ) ) ) )
                     OR
                     ( SQL_LANGUAGE_YEAR = '1992' AND
                       SQL_LANGUAGE_CONFORMANCE IN
                           ( 'ENTRY', 'INTERMEDIATE', 'FULL' ) AND
                       SQL_LANGUAGE_INTEGRITY IS NULL AND
                       ( ( SQL_LANGUAGE_BINDING  = 'DIRECT' AND
                           SQL_LANGUAGE_PROGRAMMING_LANGUGE IS NULL )
                         OR
                         ( SQL_LANGUAGE_BINDING  IN ( 'EMBEDDED', 'MODULE' )
                           AND
                           SQL_LANGUAGE_PROGRAMMING_LANGUAGE IN
                               ( 'ADA', 'C', 'COBOL',
                                 'FORTRAN', 'MUMPS', 'PASCAL', 'PLI' ) ) ) ) ) )
                 OR
                 ( SQL_LANGUAGE_SOURCE <> 'ISO 9075' )
               )
```

)

Description

1) Each row represents one binding of an ISO or implementation-defined SQL language to a standard module language, direct invocation, or an embedded programming language.

2) The value of SQL_LANGUAGE_SOURCE is the name of the source of the language definition. The source of standard SQL language is the value 'ISO 9075', while the source of an implementation-defined version of SQL is implementation-defined.

3) If the value of SQL_LANGUAGE_SOURCE is 'ISO 9075', then the value of SQL_LANGUAGE_YEAR is the year that the ISO standard was approved. Otherwise, the value of SQL_LANGUAGE_YEAR is implementation-defined.

   Note: As each new ISO SQL standard revision is approved, a new valid value of SQL_LANGUAGE_YEAR must be added to the CHECK constraint for this column.

4) If the value of SQL_LANGUAGE_SOURCE is 'ISO 9075', then the value of SQL_LANGUAGE_CONFORMANCE is the conformance level to which conformance is claimed for the ISO standard. Otherwise, the value of SQL_LANGUAGE_CONFORMANCE is implementation-defined.

5) If the value of SQL_LANGUAGE_SOURCE is 'ISO 9075' and that language contains an optional integrity enhancement feature, then the value of SQL_LANGUAGE_INTEGRITY is 'YES' if conformance is claimed to the integrity enhancement feature, and 'NO' otherwise. Otherwise, the value of SQL_LANGUAGE_INTEGRITY is implementation-defined.

6) If the value of SQL_LANGUAGE_SOURCE is 'ISO 9075', then the value of SQL_LANGUAGE_IMPLEMENTATION is null. Otherwise, the value of SQL_LANGUAGE_IMPLEMENTATION is an implementation-defined character string value.

7) If the value of SQL_LANGUAGE_SOURCE is 'ISO 9075', then the value of SQL_LANGUAGE_BINDING_STYLE is the style of binding of the SQL language. If the value of SQL_LANGUAGE_BINDING_STYLE is 'MODULE', then the binding style of <module> is supported. If the value of SQL_LANGUAGE_BINDING_STYLE is 'EMBEDDED', then the binding style of <embedded SQL host program> is supported. If the value of SQL_LANGUAGE_BINDING_STYLE is 'DIRECT', then the binding style of <direct SQL statement> is supported. Otherwise, the value of SQL_LANGUAGE_BINDING_STYLE is implementation-defined.

8) If the value of SQL_LANGUAGE_SOURCE is 'ISO 9075', then the value of SQL_LANGUAGE_PROGRAMMING_LANGUAGE is the programming language supported by the binding style indicated by the value of SQL_LANGUAGE_BINDING_STYLE. If the value of SQL_LANGUAGE_BINDING_STYLE is 'DIRECT', then SQL_LANGUAGE_PROGRAMMING_LANGUAGE is the null value. If the value of SQL_LANGUAGE_BINDING_STYLE is 'MODULE' or 'EMBEDDED', then SQL_LANGUAGE_PROGRAMMING_LANGUAGE has the value 'ADA', 'C', 'COBOL', 'FORTRAN', 'MUMPS', 'PASCAL', or 'PLI'.

   Case:

   a) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'ADA', then Ada is supported with the given binding style.

   b) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'C', then C is supported with the given binding style.

c) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'COBOL', then COBOL is supported with the given binding style.

d) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'FORTRAN', then Fortran is supported with the given binding style.

e) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'MUMPS', then MUMPS is supported with the given binding style.

f) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'PASCAL', then Pascal is supported with the given binding style.

g) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'PLI', then PL/I is supported with the given binding style.

Otherwise, the value of SQL_LANGUAGE_PROGRAMMING_LANGUAGE is implementation-defined.

23.3.38 USER_SESSION_ENABLED_ROLES base table

Function

Defines the role enabled by the <set role statement> for each SQL-session.

Definition

```
CREATE LOCAL TEMPORARY TABLE USER_SESSION_ENABLED_ROLES
       (
       USER_NAME                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       ENABLED_ROLE                    INFORMATION_SCHEMA.SQL_IDENTIFIER,

         PRIMARY KEY ( USER_NAME, ENABLED_ROLE ),

         FOREIGN KEY ( USER_NAME ) REFERENCES USERS,

         FOREIGN KEY ( ENABLED_ROLE ) REFERENCES ROLES
       )
```

Description

1) This table is a temporary table and is hence instantiated for each SQL-session. The table contains no rows when the session is first initiated.

2) When a <set role statement> is executed any existing rows are first deleted from the table and then a row is inserted.

3) USER_NAME identifies the <authorization identifier> of the current SQL-session.

4) ENABLED_ROLE identifies the <role name> enabled by the <set role statement>.

23.3.39 Temporary views

The following clauses specify temporary views on the base tables specified above.

23.3.40 ENABLED_APPLICABLE_ROLES temporary view

Function

Defines all enabled applicable roles for the current SQL-session.

Definition

```
CREATE TEMPORARY VIEW ENABLED_APPLICABLE_ROLES ( ROLE_NAME, USER_NAME )
       AS
         RECURSIVE UNION R ( ROLE_NAME, GRANTEE )
         INITIAL ( SELECT RAD.ROLE_NAME, RAD.GRANTEE FROM
                     DEFINITION_SCHEMA.USER_SESSION_ENABLED_ROLES USE,
                     DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
                   WHERE USE.ROLE_NAME = RAD.ROLE_NAME
                         AND
                         USE.USER_NAME = RAD.GRANTEE
         ITERATION ( SELECT RAD.ROLE_NAME, R.GRANTEE FROM
                       DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
                     WHERE RAD.GRANTEE = R.ROLE_NAME )
```

23.3.41 TABLE_PRIVILEGES temporary view

Function

The TABLE_PRIVILEGES view has one row for each enabled applicable table privilege for the current SQL-session plus one row for each privilege descriptor that defines a direct grant of a table privilege to any user or role. The latter rows can be used to provide the definitions of grants for which the current user is the grantor.

Definition

```
CREATE TEMPORARY VIEW TABLE_PRIVILEGES AS
       SELECT GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
              PRIVILEGE_TYPE, IS_GRANTABLE
         FROM DEFINITION_SCHEMA.PRIVILEGE_DESCRIPTORS
              WHERE COLUMN_NAME = 'ALL'
       UNION
       SELECT NULL, ER.USER_NAME,
              PD.TABLE_CATALOG, PD.TABLE_SCHEMA, PD.TABLE_NAME,
              PD.PRIVILEGE_TYPE, PD.IS_GRANTABLE
         FROM DEFINITION_SCHEMA.PRIVILEGE_DESCRIPTORS PD,
              DEFINITION_SCHEMA.ENABLED_APPLICABLE_ROLES ER
              WHERE
                PD.COLUMN_NAME = 'ALL'
              AND
                PD.GRANTEE = ER.ROLE_NAME
```

23.3.42 COLUMN_PRIVILEGES temporary view

Function

The COLUMN_PRIVILEGES view has one row for each enabled applicable column privilege for the current SQL-session plus one row for each privilege descriptor that defines a direct grant of a column privilege to any user or role in the. The latter rows can be used to provide the definitions of grants for which the current user is the grantor.

Definition

```
CREATE TEMPORARY VIEW COLUMN_PRIVILEGES AS
       SELECT GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
              COLUMN_NAME, PRIVILEGE_TYPE, IS_GRANTABLE
         FROM DEFINITION_SCHEMA.PRIVILEGE_DESCRIPTORS
         WHERE COLUMN_NAME <> 'ALL'
       UNION
       SELECT NULL, ER.USER_NAME,
              PD.TABLE_CATALOG, PD.TABLE_SCHEMA, PD.TABLE_NAME,
              PD.COLUMN_NAME, PD.PRIVILEGE_TYPE, PD.IS_GRANTABLE
         FROM DEFINITION_SCHEMA.PRIVILEGE_DESCRIPTORS PD,
              DEFINITION_SCHEMA.ENABLED_APPLICABLE_ROLES ER
         WHERE
           PD.COLUMN_NAME <> 'ALL'
         AND
           PD.GRANTEE = ER.ROLE_NAME
```

23.3.43 ROLE_TABLE_GRANTS temporary view

Function

Identifies all privileges on persistent tables that have been directly granted to the applicable roles of the current user.

Definition

```
CREATE TEMPORARY VIEW ROLE_TABLE_GRANTS AS
       SELECT GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, PRIVILEGE_TYPE
         FROM DEFINITION_SCHEMA.PRIVILEGE_DESCRIPTORS
         WHERE COLUMN_NAME = 'ALL' AND
               GRANTEE IN
                 ( RECURSIVE UNION R ( ROLE_NAME )
                   INITIAL ( SELECT ROLE_NAME FROM
                               DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
                             WHERE RAD.GRANTEE = USER
                                OR RAD.GRANTEE = 'PUBLIC' )
                   ITERATION ( SELECT ROLE_NAME FROM
                                 DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
                               WHERE RAD.GRANTEE = R.ROLE_NAME ) )
```

23.3.44 ROLE_COLUMN_GRANTS temporary view

Function

Identifies all column privileges on persistent tables that have been directly granted to an applicable role of the current user.

Definition

```
CREATE TEMPORARY VIEW ROLE_COLUMN_GRANTS AS
       SELECT GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME,
              PRIVILEGE_TYPE
         FROM DEFINITION_SCHEMA.PRIVILEGE_DESCRIPTORS
         WHERE COLUMN_NAME <> 'ALL' AND
               GRANTEE IN
                 ( RECURSIVE UNION R ( ROLE_NAME )
                   INITIAL ( SELECT ROLE_NAME FROM
                               DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
                             WHERE RAD.GRANTEE = USER
                                OR RAD.GRANTEE = 'PUBLIC' )
                   ITERATION ( SELECT ROLE_NAME FROM
                                DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
                               WHERE RAD.GRANTEE = R.ROLE_NAME ) )
```

23.3.45 USER_ROLE_GRANTS temporary view

Function

Identify roles that have been directly granted to the current user and roles granted to other users for which the current user has the WITH ADMIN OPTION.

Definition

```
CREATE TEMPORARY VIEW USER_ROLE_GRANTS AS
       SELECT GRANTEE, ROLE_NAME, IS_GRANTABLE
         FROM DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS
         WHERE GRANTEE = USER
            OR GRANTEE = 'PUBLIC'
            OR ( ROLE_NAME, 'YES' ) IN
                   ( RECURSIVE UNION R ( ROLE_NAME, IS_GRANTABLE )
                     INITIAL ( SELECT ROLE_NAME FROM
                                 DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
                               WHERE RAD.GRANTEE = USER
                                  OR RAD.GRANTEE = 'PUBLIC' )
                     ITERATION ( SELECT ROLE_NAME, IS_GRANTABLE FROM
                                   DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD

                                 WHERE RAD.GRANTEE = R.ROLE_NAME ) )
```

23.3.46 ROLE_ROLE_GRANTS temporary view

Function

Identify roles that have been directly granted to the applicable roles of the current user and roles for which the current user has the WITH ADMIN OPTION that were granted to other roles.

Definition

```
CREATE TEMPORARY VIEW ROLE_ROLE_GRANTS AS
       SELECT GRANTEE, ROLE_NAME, IS_GRANTABLE
         FROM DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS
         WHERE GRANTEE IN
            ( RECURSIVE UNION R ( ROLE_NAME )
              INITIAL
                ( SELECT ROLE_NAME FROM
                    DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
                  WHERE RAD.GRANTEE = USER
                     OR RAD.GRANTEE = 'PUBLIC' )
              ITERATION
                ( SELECT ROLE_NAME FROM
                    DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
                  WHERE RAD.GRANTEE = R.ROLE_NAME ) )
                     OR ( ROLE_NAME, 'YES' ) IN
                             ( RECURSIVE UNION R ( ROLE_NAME, IS_GRANTABLE )
                               INITIAL
                                 ( SELECT ROLE_NAME FROM
                                     DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS
                                               RAD
                                   WHERE RAD.GRANTEE = USER
                                      OR RAD.GRANTEE = 'PUBLIC' )
                               ITERATION
                                 ( SELECT ROLE_NAME, IS_GRANTABLE FROM
                                     DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS
                                               RAD
                                   WHERE RAD.GRANTEE = R.ROLE_NAME ) )
```

23.3.47 ENABLED_ROLES temporary view

****Editor's Note****

Why is this temporary view here?

It is not referenced anywhere, nor are there any privileges granted on it. Further, it was not put into the list of views at the top of this Clause.

Function

Identify all enabled applicable roles for the current user.

Definition

```
CREATE TEMPORARY VIEW ENABLED_ROLES AS
       SELECT ROLE_NAME
         FROM DEFINITION_SCHEMA.ENABLED_APPLICABLE_ROLES
           WHERE USER_NAME = USER
              OR USER_NAME = 'PUBLIC'
```

23.4 Assertions on the base tables

The following clauses specify assertions that apply to the base tables specified in Subclause 23.3, "Definition Schema".

The paramount criterion in formulating these assertions (after correctness) is ease of understanding for the human reader. There may well be formulations of the same assertions that are more efficient for some SQL-implementation, and quite possibly for all such implementations.

23.4.1 UNIQUE_CONSTRAINT_NAME assertion

Function

The UNIQUE_CONSTRAINT_NAME assertion ensures that the same combination of <schema name> and <constraint name> is not used by more than one constraint.

Note: The UNIQUE_CONSTRAINT_NAME assertion avoids the need for separate checks on DOMAINS, TABLE_CONSTRAINTS, and ASSERTIONS.

Definition

```
CREATE ASSERTION UNIQUE_CONSTRAINT_NAME
       CHECK ( 1 =
         ( SELECT MAX ( OCCURRENCES ) FROM
           ( SELECT COUNT (*) AS OCCURRENCES FROM
               ( SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                 FROM DOMAIN_CONSTRAINTS
               UNION ALL
               SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                 FROM TABLE_CONSTRAINTS
               UNION ALL
               SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                 FROM ASSERTIONS )
             GROUP BY CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
           )
         ) )
```

Description

1) The UNIQUE_CONSTRAINT_NAME assertion checks that no combination of (CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME) appears more than once in the tables DOMAINS, TABLE_CONSTRAINTS and ASSERTIONS.

23.4.2 EQUAL_KEY_DEGREES assertion

Function

The assertion EQUAL_KEY_DEGREES ensures that every foreign key is of the same degree as the corresponding unique constraint.

Definition

```
CREATE ASSERTION EQUAL_KEY_DEGREES
       CHECK
         ( NOT EXISTS
           ( SELECT * FROM       ( SELECT
                   COUNT ( DISTINCT FK.COLUMN_NAME ),
                   COUNT ( DISTINCT PK.COLUMN_NAME )
                 FROM KEY_COLUMN_USAGE AS FK,
                      REFERENTIAL_CONSTRAINTS AS RF,
                      KEY_COLUMN_USAGE AS PK
                 WHERE ( FK.CONSTRAINT_CATALOG, FK.CONSTRAINT_SCHEMA,
                           FK.CONSTRAINT_NAME )
                     = ( RF.CONSTRAINT_CATALOG, RF.CONSTRAINT_SCHEMA,
                           RF.CONSTRAINT_NAME )
                   AND
                       ( PK.CONSTRAINT_CATALOG, PK.CONSTRAINT_SCHEMA,
                           PK.CONSTRAINT_NAME )
                     = ( RF.UNIQUE_CONSTRAINT_CATALOG, RF.UNIQUE_CONSTRAINT_SCHEMA,
                           RF.UNIQUE_CONSTRAINT_NAME )
                 GROUP BY
                   RF.CONSTRAINT_CATALOG, RF.CONSTRAINT_SCHEMA, RF.CONSTRAINT_NAME
             ) AS REF ( FK_DEGREE, PK_DEGREE )
               WHERE FK_DEGREE <> PK_DEGREE ) )
```

23.4.3 KEY_DEGREE_GREATER_THAN_OR_EQUAL_TO_1 assertion

Function

The assertion KEY_DEGREE_GREATER_THAN_OR_EQUAL_TO_1 ensures that every unique or primary key constraint has at least one unique column and that every referential constraint has at least one referencing column.

Definition

```
CREATE ASSERTION KEY_DEGREE_GREATER_THAN_OR_EQUAL_TO_1
       CHECK
         ( NOT EXISTS
           ( SELECT * FROM
               TABLE_CONSTRAINTS
               FULL OUTER JOIN
                 KEY_COLUMN_USAGE
                   USING ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
               WHERE COLUMN_NAME IS NULL
                   AND CONSTRAINT_TYPE IN
                           ( 'UNIQUE', 'PRIMARY KEY', 'FOREIGN KEY' ) ) )
```

23.4.4 SUB_TABLES base table

Function

The table has one row for each supertable used to define a subtable.

Format

```
CREATE TABLE SUB_TABLES (
       TABLE_CATALOG            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA             INFORMATION_SCHEMA.SQL_IDENTIFIER,
       SUB_TABLE_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       SUPER_TABLE_NAME         INFORMATION_SCHEMA.SQL_IDENTIFIER,
       PRIMARY KEY ( TABLE_CATALOG, TABLE_SCHEMA, SUB_TABLE_NAME, SUB_TABLE_NAME ),
       FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, SUB_TABLE_NAME )
         REFERENCES TABLES,
       FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, SUPER_TABLE_NAME )
         REFERENCES TABLES )
```

Description

1) The means by which rows are inserted and deleted from this table is implementation-defined.

   Note: There is only one TABLE_SCHEMA and TABLE_SCHEMA because of the Syntax Rules in Subclause 11.3, "<table definition>".

24 Status codes

24.1 SQLSTATE

The character string value returned in an SQLSTATE parameter comprises a 2-character class value followed by a 3-character subclass value, each with an implementation-defined character set that has a one-octet form-of-use and is restricted to <digit>s and <simple Latin upper case letter>s. Table 30, "SQLSTATE class and subclass values", specifies the class value for each condition and the subclass value or values for each class value.

Class values that begin with one of the <digit>s '0', '1', '2', '3', or '4' or one of the <simple Latin upper case letter>s 'A', 'B', 'C', 'D', 'E', 'F', 'G', or 'H' are returned only for conditions defined in this
ANSI American
ISO International
Standard or in any other
ANSI American or
International Standard. The class value 'HZ' is reserved for conditions defined in ISO/IEC DIS 9579-2. Subclass values associated with such classes that also begin with one of those 13 characters are returned only for conditions defined in this
ANSI American
ISO International
Standard; subclass values associated with such classes that begin with one of the <digit>s '5', '6', '7', '8', or '9' or one of the <simple Latin upper case letter>s 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', or 'Z' are reserved for implementation-specified conditions and are called *implementation-defined subclasses*.

Class values that begin with one of the <digit>s '5', '6', '7', '8', or '9' or one of the <simple Latin upper case letter>s 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', or 'Z' are reserved for implementation-specified conditions and are called *implementation-defined classes*. All subclass values except '000', which means *no subclass*, associated with such classes are reserved for implementation-specified conditions and are called *implementation-defined subclasses*.

If a subclass value is not specified for a condition, then either subclass '000' or an implementation-defined subclass is returned.

An implementation-specified condition may duplicate a condition defined in this
ANSI American
ISO International
Standard; however, if such a condition occurs as a result of executing a statement, then the corresponding implementation-defined SQLSTATE value must not be returned in the SQLSTATE parameter but may be returned in the diagnostics area.

The ''Category'' column has the following meanings: ''S'' means that the class value given corresponds to successful completion and is a completion condition; ''W'' means that the class value given corresponds to a successful completion but with a warning and is a completion condition; ''N'' means that the class value given corresponds to a no-data situation and is a completion condition; ''X'' means that the class value given corresponds to an exception condition.

Table 30—SQLSTATE class and subclass values

Category	Condition	Class	Subcondition	Subclass
X	ambiguous cursor name	3C	(*no subclass*)	000
X	asynchronous SQL statement not accepted	05	(*no subclass*)	000
X	asynchronous SQL statement returned an SQLSTATE value	06	(*no subclass*)	000
ANSI Only–SQL3				
X	attempt to SELECT from an empty multiset	A0	(*no subclass*)	000
X	attempt to SELECT from an empty set	A1	(*no subclass*)	000
X	case not found for case statement	20	(*no subclass*)	000
X	cardinality violation	21	(*no subclass*)	000
X	connection exception	08	(*no subclass*)	000
			connection does not exist	003
			connection failure	006
			connection name in use	002
			SQL-client unable to establish SQL-connection	001
			SQL-server rejected establishment of SQL-connection	004
			transaction resolution unknown	007
X	cursor sensitivity exception	36	(*no subclass*)	000
			request failed	002
			request rejected	001
X	data exception	22	(*no subclass*)	000
			character not in repertoire	021
			datetime field overflow	008
			division by zero	012
			error in assignment	005
			indicator overflow	022
ANSI Only–SQL3				
			insertion point not unique	028

Table 30—SQLSTATE class and subclass values (Cont.)

Category	Condition	Class	Subcondition	Subclass
			interval field overflow	015
			invalid character value for cast	018
			invalid datetime format	007
			invalid enumeration name	017
			invalid enumeration value	016
			invalid escape character	019
			invalid escape sequence	025
			invalid indicator parameter value	010
			invalid limit value	020
			invalid parameter value	023
ANSI Only–SQL3				
			invalid regular expression	1B
			invalid time zone displacement value	009
			invalid update value	014
			null value, no indicator parameter	002
			numeric value out of range	003
			string data, length mismatch	026
			string data, right truncation	001
			substring error	011
			trim error	027
ISO Only–SQL3				
			unrepresentable value for USER	01A
			unterminated C string	024
X	dependent privilege descriptors still exist	2B	(*no subclass*)	000

Table 30—SQLSTATE class and subclass values (Cont.)

Category	Condition	Class	Subcondition	Subclass
X	duplicate asynchronous SQL statement identifier	41	(*no subclass*)	000
X	dynamic SQL error	07	(*no subclass*)	000
			cursor specification cannot be executed	003
			invalid descriptor count	008
			invalid descriptor index	009
			prepared statement not a cursor specification	005
			restricted data type attribute violation	006
			using clause does not match dynamic parameter specifications	001
			using clause does not match target specifications	002
			using clause required for dynamic parameters	004
			using clause required for result fields	007
ANSI Only–SQL3				
X	empty list passed to HEAD	A2	(*no subclass*)	000
X	external function call exception	39	(*no subclass*)	000
ANSI Only–SQL3				
			invalid SQLSTATE returned	001
X	external function exception	38	(*no subclass*)	000
			invalid return value for predicate	002
			not available	001
X	feature not supported	0A	(*no subclass*)	000
			multiple server transactions	001
X	integrity constraint violation	23	(*no subclass*)	000
X	invalid ADT instance	3G	(*no subclass*)	000

Table 30—SQLSTATE class and subclass values (Cont.)

Category	Condition	Class	Subcondition	Subclass
X	invalid authorization specification	28	(*no subclass*)	000
X	invalid catalog name	3D	(*no subclass*)	000
X	invalid character set name	2C	(*no subclass*)	000
X	invalid condition number	35	(*no subclass*)	000
X	invalid connection name	2E	(*no subclass*)	000
X	invalid cursor mode state	29	(*no subclass*)	000
X	invalid cursor name	34	(*no subclass*)	000
X	invalid cursor state	24	(*no subclass*)	000
X	invalid schema name	3F	(*no subclass*)	000
X	invalid SQL descriptor name	33	(*no subclass*)	000
X	invalid SQL statement name	26	(*no subclass*)	000
X	invalid SQL statement	30	(*no subclass*)	000
X	invalid target specification value	31	(*no subclass*)	000
X	invalid transaction state	25	(*no subclass*)	000
X	invalid transaction termination	2D	(*no subclass*)	000
N	no data	02	(*no subclass*)	000
X	no outstanding asynchronous SQL statement	04	(*no subclass*)	000
ANSI Only–SQL3				
X	non-single element list CAST to element type	A3	(*no subclass*)	000
X	non-single element multiset CAST to element type	A4	(*no subclass*)	000
X	non-single element set CAST to element type	A5	(*no subclass*)	000
X	Remote Database Access	HZ	(*See ISO/IEC DIS 9579-2 for the definition of protocol subconditions and subclass code values*)	
X	restrict violation	47	(*no subclass*)	000
X	savepoint exception	3B	(*no subclass*)	000
			invalid specification	001
			too many	002
X	SQL statement not yet complete	03	(*no subclass*)	000
S	successful completion	00	(*no subclass*)	000

Table 30—SQLSTATE class and subclass values (Cont.)

Category	Condition	Class	Subcondition	Subclass
X	syntax error or access rule violation	42	(*no subclass*)	000
X	syntax error or access rule violation in direct SQL statement	2A	(*no subclass*)	000
X	syntax error or access rule violation in dynamic SQL statement	37	(*no subclass*)	000
X	transaction rollback	40	(*no subclass*)	000
			integrity constraint violation	002
			serialization failure	001
			statement completion unknown	003
			triggered action exception	004
X	triggered action exception	09	(*no subclass*)	000
X	triggered data change violation	27	(*no subclass*)	000
ANSI Only–SQL3				
X	unable to determine element type for LIST	A6	(*no subclass*)	000
X	unable to determine element type for MULTISET	A7	(*no subclass*)	000
X	unable to determine element type for SET	A8	(*no subclass*)	000
W	warning	01	(*no subclass*)	000
			cursor operation conflict	001
			disconnect error	002
ANSI Only–SQL3				
			external routine warning (the value of *xx* to be chosen by the author of the external routine)	H*xx*
			implicit zero-bit padding	008
			insufficient item descriptor areas	005
			null value eliminated in set function	003
			privilege not granted	007
			privilege not revoked	006

Table 30—SQLSTATE class and subclass values (Cont.)

Category	Condition	Class	Subcondition	Subclass
			query expression too long for information schema	00A
			search condition too long for information schema	009
			string data, right truncation	004
X	with check option violation	44	(*no subclass*)	000

24.2 SQLCODE

Table 31, "SQLCODE values", specifies the integer value returned in an SQLCODE parameter for each condition. The negative values that indicate exception conditions are implementation-defined.

Table 31—SQLCODE values

Value	Condition
0	successful completion
+100	no data
−*n*	exception

Note: SQLSTATE is the preferred status parameter. The SQLCODE status parameter is a deprecated feature that is supported for compatibility with earlier versions of this
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

25 Conformance

25.1 Introduction

This
ANSI American
ISO International
Standard specifies conforming SQL language and conforming SQL-implementations.

Conforming SQL language shall abide by the BNF Format, associated Syntax Rules and Access Rules, definitions, and descriptions.

A conforming SQL-implementation shall process conforming SQL language according to the associated General Rules, definitions, and descriptions.

The object identifier for Database Language SQL is specified in Subclause 3.4, "Object identifier for Database Language SQL".

25.2 Claims of conformance

Claims of conformance to this
ANSI American
ISO International
Standard shall state:

1) Which *level of conformance* is claimed:

   a) Full SQL (The complete database language specified in this
      ANSI American
      ISO International
      Standard.)

   b) Intermediate SQL (Intermediate SQL is a subset of Full SQL as specified in the Leveling Rules.)

   c) Entry SQL (Entry SQL is a subset of Intermediate SQL as specified in the Leveling Rules.)

2) Which of the following *binding styles* are supported:

   a) Module (<module>)

   b) Embedded syntax (<embedded SQL host program>)

   c) Direct invocation and processing of SQL language (<direct SQL statement>)

3) For the binding styles module or embedded syntax, which of the following programming languages are supported:

   a) Ada

   b) C

c) COBOL

d) Fortran

e) MUMPS

f) Pascal

g) PL/I

4) The definitions for all elements and actions that this Standard specifies as implementation-defined.

25.3 Extensions and options

A conforming implementation may provide additional facilities or options not specified by this
ANSI American
ISO International
Standard. This may imply an implementation-defined extension of the list of reserved words (<reserved word>) and thereby may prevent proper processing of some programs that otherwise meet the requirements of this
ANSI American
ISO International
Standard.

An implementation remains conforming even if it provides user options to process nonconforming SQL language or to process conforming SQL language in a nonconforming manner.

25.4 Flagger requirements

Implementations that claim conformance only to Entry SQL may, but are not required to, provide an SQL Flagger (see Subclause 4.43, "SQL Flagger").

Implementations that claim conformance to Intermediate SQL shall provide an SQL Flagger (see Subclause 4.43, "SQL Flagger") that supports the following ''level of flagging'' options:

— Entry SQL Flagging

— Intermediate SQL Flagging

and the following ''extent of checking'' option:

— Syntax Only

Implementations that claim conformance to Full SQL shall provide an SQL Flagger (see Subclause 4.43, "SQL Flagger") that supports the following ''level of flagging'' options:

— Entry SQL Flagging

— Intermediate SQL Flagging

— Full SQL Flagging

and the following ‘‘extent of checking’’ options:

— Syntax Only

— Catalog Lookup

25.5 Processing methods

This
ANSI American
ISO International
Standard does not define the method by which an <embedded SQL host program> is processed. Although the processing of <embedded SQL host program> is defined in terms of derivation of a program compliant with a programming language standard and a <module>, implementations of SQL are not constrained to follow that method, provided that effect is achieved.

Although the processing of <direct SQL statement> is defined in terms of calls to <routine>s in a <module>, implementations of Direct Invocation are not constrained to follow that method, so long as the same effect is achieved.

Annex A
(Informative)

Leveling the SQL Language

This Annex describes the restrictions placed on conforming Full SQL, Intermediate SQL and Entry SQL language.

A.1 Full SQL Specifications

1) Subclause 5.3, "<literal>":

   a) A <general literal> shall not be an <enumeration literal>.

   b) A <general literal> shall not be a <boolean literal>.

   ANSI Only–SQL3

   c) A <general literal> shall not be an <oid literal>.

2) Subclause 5.4, "Names and identifiers":

   a) A <column name> shall not specify OID.

3) Subclause 6.1, "<data type>":

   a) A <datatype> shall not be an <object identifier type>.

4) Subclause 6.4, "<attribute reference>":

   a) Conforming Full SQL language shall contain no <attribute reference>.

5) Subclause 6.14, "<value expression>":

   a) A <value expression> shall not be an <enumerated value expression>.

   b) A <value expression> shall not be a <boolean value expression>.

6) Subclause 6.17, "<enumerated value expression>":

   a) Conforming Full SQL language shall contain no <enumerated value expression>.

7) Subclause 6.21, "<boolean value expression>":

   a) Conforming Full SQL language shall contain no <boolean value expression>.

8) Subclause 7.1, "<row value constructor>":

   a) A <null specification> shall contain no <null state>.

9) Subclause 7.8, "<joined table>":

   a) A <qualified join> shall contain no <constraint join>.

10) Subclause 7.12, "<query specification>":

   a) A <query specification> shall contain no <user-defined updatability clause> or <referencing new values clause>.

   b) A <derived column> shall contain no <user-defined column update clause>.

   c) A query specification is not updatable if other than a <column reference> or if a <column reference> appears more than once.

11) Subclause 7.13, "<query expression>":

   a) A <non-join query term> shall contain no <recursive union>.

12) Subclause 7.14, "<recursive union>":

   a) Conforming Full SQL language shall contain no <recursive union>.

13) Subclause 8.1, "<predicate>":

   a) Conforming Full SQL language shall contain no <similar predicate>.

   b) Conforming Full SQL language shall contain no <quantified predicate>.

   c) Conforming Full SQL language shall contain no <there is predicate>.

   d) Conforming Full SQL language shall contain no <function invocation predicate>.

14) Subclause 8.6, "<similar predicate>":

   a) Conforming Full SQL language shall contain no <similar predicate>.

15) Subclause 8.7, "<null predicate>":

   a) A <null predicate> shall contain no <null values specification>.

16) Subclause 8.13, "<quantified predicate>":

   a) Conforming Full SQL language shall contain no <quantified predicate>.

17) Subclause 8.14, "<there is predicate>":

   a) Conforming Full SQL language shall contain no <there is predicate>.

18) Subclause 8.17, "<search condition>":

a) A <boolean primary> shall not specify a <boolean value expression>.

19) Subclause 10.3, "<routine invocation>":

a) Conforming Full SQL language shall contain no <external function invocation>.

20) Subclause 10.3, "<routine invocation>":

a) Conforming Full SQL language shall contain no <SQL function invocation>.

21) Subclause 10.5, "<privileges>":

a) An <action> shall not specify EXECUTE, TRIGGER, or UNDER.

b) An <action> that specifies SELECT shall not contain a <privilege column list>.

22) Subclause 11.1, "<schema definition>":

a) A <schema element> shall not be a <null class definition>.

b) A <schema element> shall not be a <trigger definition>.

c) A <schema element> shall not be an <external function declaration>.

d) A <schema element> shall not be a <routine>.

e) A <schema element> shall not be an <abstract data type definition>.

f) A <schema element> shall not be a <module definition>.

g) A <schema element> shall not be a <role definition>.

h) A <schema element> shall not be a <grant role statement>.

23) Subclause 11.3, "<table definition>":

a) A <table source> shall not be a <new object table definition>.

b) A <table source> shall not be an <existing object table definition>.

c) A <table source> shall not be a <subtable definition>.

d) A <base table definition> shall not contain a <new type specification>.

e) A <table element> shall not be a <function declaration>.

f) A <table element> shall not be an <equals clause>.

g) A <table element> shall not be a <less-than clause>.

h) A <table element> shall not be a <like clause>.

24) Subclause 11.4, "<column definition>":

a) A <column definition> shall not contain an <encapsulation level>.

b) A <column definition> shall not contain a <null clause>.

25) Subclause 11.5, "<default clause>":

a) A <default option> that specifies NULL shall not contain a <null state>.

b) A <default option> shall not be a <function invocation>.

26) Subclause 11.7, "<unique constraint definition>":

a) A <unique constraint definition> shall specify a <unique column list>.

27) Subclause 11.8, "<referential constraint definition>":

a) A <references specification> shall not contain PENDANT.

b) If a <referenced table and columns> specifies a <left paren>, then it shall also specify a <reference column list>.

c) A <referential action> shall not be RESTRICT.

d) A <referential action> shall not be NO ACTION.

e) The data type of each referencing column shall be the same as the data type of the corresponding referenced column.

28) Subclause 11.10, "<alter table statement>":

a) An <alter table action> shall not be a <add supertable clause>.

b) An <alter table action> shall not be a <drop supertable clause>.

29) Subclause 11.17, "<add supertable clause>":

a) Conforming Full SQL language shall contain no <add supertable clause>.

30) Subclause 11.18, "<drop supertable clause>":

a) Conforming Full SQL language shall contain no <drop supertable clause>.

31) Subclause 11.31, "<null class definition>":

a) Conforming Full SQL language shall contain no <null class definition>.

32) Subclause 11.32, "<drop null class statement>":

a) Conforming Full SQL language shall contain no <drop null class statement>.

33) Subclause 11.33, "<character set definition>":

a) A <character set source> shall not contain a <plus sign>.

b) A <character set source> shall not contain a <character list>.

34) Subclause 11.35, "<collation definition>":

a) A <collation source> shall not be a <collation dictionary specification>.

b) A <collation source> shall not be a <collation routine specification>.

c) A <collation sequence definition> shall not be a <internal collation source>.

35) Subclause 11.36, "<drop collation statement>":

a) A <drop behavior> shall not specify CASCADE.

36) Subclause 11.37, "<translation definition>":

a) A <translation source> shall not be a <translation routine>.

b) A <translation specification> shall not be a <internal translation source>.

37) Subclause 11.39, "<assertion definition>":

a) An <assertion definition> shall contain no <assertion trigger>.

b) A <triggered assertion> shall not specify FOR.

38) Subclause 11.41, "<trigger definition>":

a) Conforming Full SQL language shall contain no <trigger definition>.

39) Subclause 11.42, "<drop trigger statement>":

a) Conforming Full SQL language shall contain no <drop trigger statement>.

40) Subclause 11.43, "<drop routine statement>":

a) Conforming Full SQL language shall contain no <drop routine statement>.

41) Subclause 11.44, "<abstract data type definition>":

a) Conforming Full SQL language shall contain no <abstract data type definition>.

42) Subclause 11.49, "<drop data type statement>":

a) Conforming Full SQL language shall contain no <drop data type statement>.

43) Subclause 11.50, "<module definition>":

a) Conforming Full SQL language shall contain no <module definition>.

44) Subclause 11.51, "<drop module statement>":

a) Conforming Full SQL language shall contain no <drop module statement>.

45) Subclause 11.52, "<grant statement>":

a) An <object name> shall not specify SET or LIST.

46) Subclause 11.53, "<role definition>":

a) Conforming Full SQL language shall contain no <role definition>.

47) Subclause 11.54, "<grant role statement>":

a) Conforming Full SQL language shall contain no <grant role statement>.

48) Subclause 11.55, "<revoke role statement>":

a) Conforming Full SQL language shall contain no <revoke role statement>.

49) Subclause 11.56, "<drop role statement>":

a) Conforming Full SQL language shall contain no <drop role statement>.

50) Subclause 13.1, "<module>":

a) A <module> shall not contain a <temporary table declaration>.

51) Subclause 13.3, "<SQL procedure statement>":

a) An <SQL procedure statement> shall not specify ASYNC.

b) An <SQL executable statement> shall not be a <SQL control statement>.

c) An <SQL schema definition statement> shall not be a <role definition>, <grant role statement>, <null class definition>, <trigger definition>, <routine>, <abstract data type definition>, <type template definition>, or <module definition>.

d) An <SQL shema manipulation statement> shall not be a <revoke role statement>, <drop role statement>, <drop null class statement>, <drop trigger statement>, <drop routine statement>, <drop data type statement>, <drop type template statement>, or <drop module statement>.

e) An <SQL transaction statement> shall not be a <test completion statement>, <savepoint statement>, or <release savepoint statement>.

f) An <SQL session statement> shall not be a <set role statement>.

g) An <SQL diagnostics statement> shall not be a <signal statement> or <resignal statement>.

52) Subclause 14.1, "<call statement>":

a) Conforming Full SQL language shall contain no <call statement>.

53) Subclause 14.2, "<return statement>":

a) Conforming Full SQL language shall contain no <return statement>.

54) Subclause 14.3, "<compound statement>":

a) Conforming Full SQL language shall contain no <compound statement>.

55) Subclause 14.4, "<exception handler>":

a) Conforming Full SQL language shall contain no <exception handler>.

56) Subclause 14.5, "<exception declaration>":

a) Conforming Full SQL language shall contain no <exception declaration>.

57) Subclause 14.6, "<SQL variable declaration>":

a) Conforming Full SQL language shall contain no <SQL variable declaration>.

58) Subclause 14.10, "<case statement>":

a) Conforming Full SQL language shall contain no <case statement>.

59) Subclause 14.11, "<if statement>":

a) Conforming Full SQL language shall contain no <if statement>.

60) Subclause 14.12, "<leave statement>":

a) Conforming Full SQL language shall contain no <leave statement>.

61) Subclause 14.13, "<loop statement>":

a) Conforming Full SQL language shall contain no <loop statement>.

62) Subclause 14.14, "<for statement":

a) Conforming Full SQL language shall contain no <for statement>.

63) Subclause 14.15, "<signal statement>":

a) Conforming Full SQL language shall contain no <signal statement>.

64) Subclause 14.16, "<resignal statement>":

a) Conforming Full SQL language shall contain no <resignal statement>.

65) Subclause 14.7, "<assignment statement>":

a) Conforming Full SQL language shall contain no <assignment statement>.

66) Subclause 14.8, "<new statement>":

a) Conforming Full SQL language shall contain no <new statement>.

67) Subclause 14.9, "<destroy statement>":

a) Conforming Full SQL language shall contain no <destroy statement>.

68) Subclause 15.1, "<declare cursor>":

a) A <declare cursor> shall not specify SENSITIVE.

69) Subclause 15.2, "<open statement>":

a) An <open statement> shall not contain an <open cascade option>.

70) Subclause 15.8, "<insert statement>":

a) An <insert statement> shall not specify OBJECT or ROW.

b) An <insert statement> shall not specify ALIAS.

71) Subclause 15.9, "<update statement: positioned>":

a) An <update statement: positioned> shall contain a <table name>.

b) An <update statement: positioned> shall not contain an <update type>.

c) An <update target> shall not contain an <object column list>.

72) Subclause 15.10, "<update statement: searched>":

a) An <update statement: searched> shall not contain an <update type>.

b) An <update target> shall not contain an <object column list>.

73) Subclause 15.12, "<temporary view definition>":

a) Conforming Full SQL language shall contain no <temporary view definition>.

74) Subclause 16.3, "<test completion statement>":

a) Conforming Full SQL language shall contain no <test completion statement>.

75) Subclause 16.4, "<savepoint statement>":

a) Conforming Full SQL language shall contain no <savepoint statement>.

76) Subclause 16.5, "<release savepoint statement>":

a) Conforming Full SQL language shall contain no <release savepoint statement>.

77) Subclause 16.7, "<rollback statement>":

a) A <rollback statement> shall contain no <savepoint clause>.

78) Subclause 18.5, "<set role statement>":

a) Conforming Full SQL language shall contain no <set role statement>.

79) Subclause 21.1, "<embedded SQL host program>":

a) An <embedded SQL statement> shall contain no <temporary view definition>.

b) An <embedded SQL statement> shall contain no <external function declaration>.

80) Subclause 21.2, "<embedded exception declaration>":

a) A <SQL condition> shall not specify SQLSTATE or CONSTRAINT.

b) A <major category> shall not specify SQLEXCEPTION or SQLWARNING.

A.2 Intermediate SQL Specifications

1) Subclause 5.2, "<token> and <separator>":

a) In conforming Intermediate SQL language, a <regular identifier> or a <delimited identifier body> shall not comprise more than 18 <character representation>s.

b) A <delimiter token> shall not be a <bit string literal> or a <hex string literal>.

c) Conforming Intermediate SQL language shall contain no <identifier body> that ends in an <underscore>.

2) Subclause 5.3, "<literal>":

   a) An <unsigned integer> that is a <seconds fraction> shall not contain more than 6 <digit>s.

   b) A <literal> or <unsigned literal> shall not be a <bit string literal> or a <hex string literal>.

   c) A <character string literal> shall not specify a <character set specification>.

3) Subclause 5.4, "Names and identifiers":

   a) Conforming Intermediate SQL language shall not contain any <extended statement name> or <extended cursor name>.

   b) Conforming Intermediate SQL language shall not contain any <collation name>.

   c) Conforming Intermediate SQL language shall not contain any explicit <catalog name>.

   d) Conforming Intermediate SQL language shall not contain any <translation name> or <form-of-use conversion name>.

   e) An <identifier> shall not specify a <character set specification>.

4) Subclause 6.1, "<data type>":

   a) A <datetime type> shall not specify a <time precision> or <timestamp precision>.

   b) A <data type> shall not be a <bit string type>.

5) Subclause 6.2, "<value specification> and <target specification>":

   a) In Intermediate SQL, the specific data type of <indicator parameter>s and <indicator variable>s shall be the same implementation-defined data type.

6) Subclause 6.5, "<table reference>":

   a) A <table reference> shall not be a <derived table>.

7) Subclause 6.7, "<set function specification>":

   a) If a <general set function> specifies DISTINCT, then the <value expression> shall be a <column reference>.

8) Subclause 6.8, "<numeric value function>":

   a) A <numeric value function> shall not be a <position expression>.

   b) A <numeric value function> shall not contain a <length expression> that is a <bit length expression>.

9) Subclause 6.9, "<string value function>":

   a) A <character value function> shall not be a <fold>.

   b) Conforming Intermediate SQL language shall contain no <character translation>.

   c) Conforming Intermediate SQL language shall contain no <form-of-use conversion>.

   d) Conforming Intermediate SQL language shall contain no <bit value function>.

10) Subclause 6.10, "<datetime value function>":

a) Conforming Intermediate SQL language shall contain no <time precision> or <timestamp precision>.

11) Subclause 6.16, "<string value expression>":

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

b) Conforming Intermediate SQL language shall contain no <bit value expression>.

12) Subclause 7.1, "<row value constructor>":

a) A <row value constructor> that is not simply contained in a <table value constructor> shall not contain more than one <row value constructor element>.

13) Subclause 7.2, "<table value constructor>":

a) A <table value constructor> shall contain exactly one <row value constructor> that shall be of the form ''(<value expression list>)''.

b) A <table value constructor> shall be the <query expression> of an <insert statement>.

14) Subclause 7.6, "<table expression>":

a) If the table identified in the <from clause> is a grouped view, then the <table expression> shall not contain a <where clause>, <group by clause>, or <having clause>.

15) Subclause 7.7, "<from clause>":

a) A <table reference> shall not be a <derived table>.

b) If the table identified by <table name> is a grouped view, then the <from clause> shall contain exactly one <table reference>.

16) Subclause 7.8, "<joined table>":

a) Conforming Intermediate SQL language shall contain no <cross join>.

b) Conforming Intermediate SQL language shall not specify UNION JOIN.

17) Subclause 7.10, "<group by clause>":

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

18) Subclause 7.12, "<query specification>":

a) The <key word> DISTINCT shall not be specified more than once in a <query specification>, excluding any <subquery> of that <query specification>.

b) If the <table expression> of the <query specification> is a grouped view, then the <select list> shall not contain a <set function specification>.

19) Subclause 7.13, "<query expression>":

a) A <simple table> shall not be a <table value constructor> except in an <insert statement>.

b) Conforming Intermediate SQL shall contain no <explicit table>.

20) Subclause 8.1, "<predicate>":

a) A <predicate> shall not be a <match predicate>.

21) Subclause 8.4, "<in predicate>":

a) Conforming Intermediate SQL language shall not contain a <value expression> in an <in value list> that is not a <value specification>.

22) Subclause 8.11, "<match predicate>":

a) Conforming Intermediate SQL language shall not contain any <match predicate>.

23) Subclause 8.17, "<search condition>":

a) A <boolean test> shall not specify a <truth value>.

24) Subclause 10.5, "<privileges>":

a) An <action> that specifies INSERT shall not contain a <privilege column list>.

25) Subclause 10.7, "<collate clause>":

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

26) Subclause 11.1, "<schema definition>":

a) A <schema element> shall not be an <assertion definition>.

b) Conforming Intermediate SQL language shall not contain any <collation definition>.

c) Conforming Intermediate SQL language shall not contain any <translation definition>.

27) Subclause 11.3, "<table definition>":

a) Conforming Intermediate SQL language shall not specify GLOBAL TEMPORARY and shall not reference any global temporary table.

28) Subclause 11.4, "<column definition>":

a) A <constraint attributes> shall not be specified, but INITIALLY IMMEDIATE and NOT DEFERRABLE are implicit.

b) Conforming Intermediate SQL language shall not contain any <collate clause>.

29) Subclause 11.6, "<table constraint definition>":

a) A <constraint attributes> shall not be specified, but INITIALLY IMMEDIATE and NOT DEFERRABLE are implicit.

30) Subclause 11.8, "<referential constraint definition>":

a) A <references specification> shall not specify MATCH.

b) A <referential triggered action> shall not contain an <update rule>.

31) Subclause 11.9, "<check constraint definition>":

a) The <search condition> contained in a <check constraint definition> shall not contain a <subquery>.

b) The REFERENCES privilege is not required for <check constraint definition> access.

32) Subclause 11.22, "<view definition>":

a) Conforming Intermediate SQL shall not contain any <levels clause>, but the effect shall be that defined for a <levels clause> of CASCADED.

33) Subclause 11.24, "<domain definition>":

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

34) Subclause 11.25, "<alter domain statement>":

a) Conforming Intermediate SQL language shall contain no <alter domain statement>.

35) Subclause 11.26, "<set domain default clause>":

a) Conforming Intermediate SQL language shall contain no <set domain default clause>.

36) Subclause 11.27, "<drop domain default clause>":

a) Conforming Intermediate SQL language shall contain no <drop domain default clause>.

37) Subclause 11.28, "<add domain constraint definition>":

a) Conforming Intermediate SQL language shall contain no <add domain constraint definition>.

38) Subclause 11.29, "<drop domain constraint definition>":

a) Conforming Intermediate SQL language shall contain no <drop domain constraint definition>.

39) Subclause 11.35, "<collation definition>":

a) Conforming Intermediate SQL language shall not contain any <collation definition>.

40) Subclause 11.36, "<drop collation statement>":

a) Conforming Intermediate SQL language shall not contain any <drop collation statement>.

41) Subclause 11.37, "<translation definition>":

a) Conforming Intermediate SQL language shall contain no <translation definition>.

42) Subclause 11.38, "<drop translation statement>":

a) Conforming Intermediate SQL language shall contain no <drop translation statement>.

43) Subclause 11.39, "<assertion definition>":

a) Conforming Intermediate SQL language shall not contain any <assertion definition>.

44) Subclause 11.40, "<drop assertion statement>":

a) Conforming Intermediate SQL language shall not contain any <drop assertion statement>.

45) Subclause 11.52, "<grant statement>":

a) In Conforming Intermediate SQL language, an <object name> shall not specify COLLATION or TRANSLATION.

46) Subclause 13.1, "<module>":

a) A <module> shall not contain a <temporary table declaration>.

47) Subclause 13.2, "<routine>":

a) A <parameter declaration> shall not specify a <data type> that is BIT or BIT VARYING.

48) Subclause 13.3, "<SQL procedure statement>":

a) An <SQL transaction statement> shall not be a <set constraints mode statement>.

b) An <SQL schema statement> shall not be an <assertion definition>.

c) An <SQL schema statement> shall not be a <drop assertion statement>.

d) An <SQL dynamic statement> shall not be an <allocate cursor statement> statement.

e) An <SQL session statement> shall not be a <set schema statement>, a <set catalog statement>, or a <set names statement>.

f) An <SQL schema statement> shall not be a <collation definition>, a <drop collation statement>, a <translation definition>, or a <drop translation statement>.

g) An <SQL dynamic statement shall not be a <deallocate prepared statement>, or a <describe input statement>.

h) An <SQL procedure statement> shall not be an <SQL connection statement>.

49) Subclause 15.1, "<declare cursor>":

a) Conforming Intermediate SQL language shall not specify INSENSITIVE.

b) If an <updatability clause> of FOR UPDATE with or without a <column name list> is specified, then neither SCROLL nor ORDER BY shall be specified.

50) Subclause 15.3, "<fetch statement>":

a) A <fetch statement> shall not contain a <fetch orientation>.

51) Subclause 15.5, "<select statement: single row>":

a) The <table expression> shall not include a <group by clause> or a <having clause> and shall not identify a grouped view.

52) Subclause 15.6, "<delete statement: positioned>":

a) Let *T* be the table identified by the <table name>. *T* shall not be a table that is identified in the <from clause> of any <subquery> contained in the <cursor specification> referenced by the <cursor name>.

53) Subclause 15.7, "<delete statement: searched>":

a) Let *T* be the table identified by the <table name>. *T* shall not be a table that is identified in the <from clause> of any <subquery> contained in the <search condition>.

54) Subclause 15.8, "<insert statement>":

a) The base table that would be changed as a result of executing the <insert statement> shall not be identified in any <from clause> generally contained in the <query expression>.

55) Subclause 15.9, "<update statement: positioned>":

a) Let *T* be the table identified by the <table name>. *T* shall not be a table that is identified in the <from clause> of any <subquery> contained in the <cursor specification> referenced by the <cursor name>.

56) Subclause 15.10, "<update statement: searched>":

a) Let *T* be the table identified by the <table name>. *T* shall not be a table that is identified in the <from clause> of any <subquery> contained in the <search condition> or contained in the <value expression> of the <set clause>.

57) Subclause 15.11, "<temporary table declaration>":

a) Conforming Intermediate SQL language shall not contain any <temporary table declaration>.

58) Subclause 16.2, "<set constraints mode statement>":

a) Conforming Intermediate SQL language shall not contain any <set constraints mode statement>.

59) Subclause 17.1, "<connect statement>":

a) Conforming Intermediate SQL language shall not contain any <connect statement>.

60) Subclause 17.2, "<set connection statement>":

a) Conforming Intermediate SQL language shall not contain any <set connection statement>.

61) Subclause 17.3, "<disconnect statement>":

a) Conforming Intermediate SQL language shall not contain any <disconnect statement>.

62) Subclause 18.1, "<set catalog statement>":

a) Conforming Intermediate SQL language shall not contain any <set catalog statement>.

63) Subclause 18.2, "<set schema statement>":

a) Conforming Intermediate SQL language shall not contain any <set schema statement>.

64) Subclause 18.3, "<set names statement>":

a) Conforming Intermediate SQL language shall not contain any <set names statement>.

65) Subclause 19.2, "<allocate descriptor statement>":

a) An <occurrences> and a <descriptor name> shall be a <literal>.

66) Subclause 19.3, "<deallocate descriptor statement>":

a) A <descriptor name> shall be a <literal>.

67) Subclause 19.4, "<get descriptor statement>":

a) A <descriptor name> shall be a <literal>.

68) Subclause 19.5, "<set descriptor statement>":

a) A <descriptor name> shall be a <literal>.

69) Subclause 19.6, "<prepare statement>":

a) A <preparable SQL schema statement> shall not be an <assertion definition> or a <drop assertion statement>.

b) A <preparable SQL schema statement> shall not be a <collation definition>, a <drop collation statement>, a <translation definition>, or a <drop translation statement>.

c) A <preparable SQL session statement> shall not be a <set schema statement>.

d) A <preparable SQL session statement> shall not be a <set catalog statement>.

e) A <preparable SQL session statement> shall not be a <set names statement>.

f) A <preparable SQL transaction statement> shall not be a <set constraints mode statement>.

g) A <preparable SQL data statement> shall not be a <dynamic single row select statement>.

70) Subclause 19.7, "<deallocate prepared statement>":

a) Conforming Intermediate SQL language shall contain no <deallocate prepared statement>.

71) Subclause 19.8, "<describe statement>":

a) Conforming Intermediate SQL language shall not contain any <describe input statement>.

72) Subclause 19.9, "<using clause>":

a) A <descriptor name> shall be a <literal>.

73) Subclause 19.10, "<execute statement>":

a) Conforming Intermediate SQL language shall contain no <result using clause>.

74) Subclause 19.12, "<dynamic declare cursor>":

a) Conforming Intermediate SQL language shall contain no <dynamic declare cursor> that specifies INSENSITIVE.

b) If an <updatability clause> of FOR UPDATE with or without a <column name list> is specified, then neither SCROLL nor ORDER BY shall be specified.

75) Subclause 19.13, "<allocate cursor statement>":

a) Conforming Intermediate SQL language shall not contain any <allocate cursor statement>.

76) Subclause 19.15, "<dynamic fetch statement>":

a) A <dynamic fetch statement> shall not contain a <fetch orientation>.

77) Subclause 19.17, "<dynamic delete statement: positioned>":

a) Let *T* be the table identified by the <table name>. *T* shall not be a table that is identified in the <from clause> of any <subquery> contained in the <cursor specification> referenced by the <dynamic cursor name>.

78) Subclause 19.18, "<dynamic update statement: positioned>":

a) Let *T* be the table identified by the <table name>. *T* shall not be a table that is identified in the <from clause> of any <subquery> contained in the <cursor specification> referenced by the <dynamic cursor name>.

79) Subclause 19.19, "<preparable dynamic delete statement: positioned>":

a) Conforming Intermediate SQL language shall contain no <preparable dynamic delete statement: positioned>.

80) Subclause 19.20, "<preparable dynamic update statement: positioned>":

a) Conforming Intermediate SQL language shall contain no <preparable dynamic update statement: positioned>.

81) Subclause 21.1, "<embedded SQL host program>":

a) An <embedded SQL statement> shall not contain a <temporary table declaration>.

82) Subclause 21.3, "<embedded SQL Ada program>":

a) An <Ada variable definition> shall not specify a bit string variable.

83) Subclause 21.4, "<embedded SQL C program>":

a) A <C derived variable> shall not be a <C bit variable>.

84) Subclause 21.5, "<embedded SQL COBOL program>":

a) A <COBOL type specification> shall not be a <COBOL bit type>.

85) Subclause 21.6, "<embedded SQL Fortran program>":

a) A <Fortran type specification> shall not specify BIT.

86) Subclause 21.8, "<embedded SQL Pascal program>":

a) A <Pascal type specification> shall not specify BIT or PACKED ARRAY [1..<length>] OF BIT.

87) Subclause 21.9, "<embedded SQL PL/I program>":

a) A <PL/I type specification> shall not specify BIT or BIT VARYING.

88) Subclause 22.1, "<direct SQL statement>":

a) A <direct SQL data statement> shall not be a <temporary table declaration>.

b) An <SQL session statement> shall not be a <set schema statement>, a <set catalog statement>, or a <set names statement>.

c) An <SQL transaction statement> shall not be a <set constraints mode statement>.

d) An <SQL schema statement> shall not be an <assertion definition> or a <drop assertion statement>.

e) An <SQL schema statement> shall not be a <collation definition>, a <drop collation statement>, a <translation definition> or a <drop translation statement>.

f) A <direct SQL statement> shall not be an <SQL connection statement>.

89) Subclause 23.2, "Information Schema":

a) Conforming Intermediate SQL language shall reference only the following Information Schema tables: SCHEMATA, DOMAINS, TABLES, VIEWS, COLUMNS, TABLE_PRIVILEGES, COLUMN_PRIVILEGES, USAGE_PRIVILEGES, REFERENTIAL_CONSTRAINTS, CHECK_CONSTRAINTS, KEY_COLUMN_USAGE, ASSERTIONS, CHARACTER_SETS, COLLATIONS, VIEW_COLUMN_USAGE, CONSTRAINT_TABLE_USAGE, CONSTRAINT_COLUMN_USAGE, SQL_LANGUAGES, COLUMN_DOMAIN_USAGE, and VIEW_TABLE_USAGE.

A.3 Entry SQL Specifications

1) All Intermediate SQL specifications are included as Entry SQL specifications.

2) Subclause 5.2, "<token> and <separator>":

a) A <delimiter token> shall not be a <national character string literal>.

b) An <identifier body> shall contain no <simple Latin lower case letter>.

3) Subclause 5.3, "<literal>":

a) A <literal> or <unsigned literal> shall not be a <national character string literal>.

b) A <literal> or <unsigned literal> shall not be a <datetime literal> or an <interval literal>.

4) Subclause 5.4, "Names and identifiers":

a) Conforming Entry SQL language shall not contain any <domain name>, <SQL statement name>, <dynamic cursor name>, or <constraint name>.

5) Subclause 6.1, "<data type>":

a) A <character string type> shall not specify VARYING or VARCHAR.

b) A <data type> shall not be a <datetime type> or an <interval type>.

c) A <data type> shall not be a <national character string type> nor specify CHARACTER SET.

6) Subclause 6.2, "<value specification> and <target specification>":

a) A <general value specification> shall not be a <dynamic parameter specification>.

b) A <general value specification> shall not specify VALUE.

c) A <general value specification> shall not specify SESSION_USER, SYSTEM_USER, or CURRENT_USER.

7) Subclause 6.5, "<table reference>":

a) A <table reference> shall not be a <joined table>.

b) The optional <key word> AS shall not be specified.

c) <derived column list> shall not be specified.

8) Subclause 6.7, "<set function specification>":

a) If a <general set function> specifies or implies ALL, then COUNT shall not be specified.

b) If a <general set function> specifies or implies ALL, then the <value expression> shall include a <column reference> that references a column of *T*.

c) If the <value expression> contains a <column reference> that is an outer reference, then the <value expression> shall be a <column reference>.

d) No <column reference> contained in a <set function specification> shall reference a column derived from a <value expression> that generally contains a <set function specification>.

9) Subclause 6.8, "<numeric value function>":

a) A <numeric value function> shall not be a <length expression>.

b) A <numeric value function> shall not be an <extract expression>.

10) Subclause 6.9, "<string value function>":

a) A <character value function> shall not be a <character substring function>.

b) A <character value function> shall not be a <trim function>.

11) Subclause 6.10, "<datetime value function>":

a) Conforming Entry SQL language shall not contain any <datetime value function>.

12) Subclause 6.12, "<case expression>":

a) Conforming Entry SQL language shall not contain any <case expression>.

13) Subclause 6.13, "<cast specification>":

a) Conforming Entry SQL language shall not contain any <cast specification>.

14) Subclause 6.14, "<value expression>":

a) A <value expression> shall not be a <datetime value expression>.

b) A <value expression> shall not be an <interval value expression>.

c) A <value expression primary> shall not be a <case expression>.

d) A <value expression primary> shall not be a <cast specification>.

e) A <value expression primary> shall not be a <scalar subquery> except when the <value expression primary> is simply contained in a <value expression> that is simply contained in the second <row value constructor> of a <comparison predicate>.

15) Subclause 6.16, "<string value expression>":

a) A <character value expression> shall not be a <concatenation>.

16) Subclause 6.18, "<datetime value expression>":

a) Conforming Entry SQL language shall not contain any <datetime value expression>.

17) Subclause 6.19, "<interval value expression>":

a) Conforming Entry SQL language shall not contain any <interval value expression>.

18) Subclause 7.1, "<row value constructor>":

a) A <value expression> that is contained in the <row value constructor> that is contained in the <table value constructor> that is contained in the <query expression> of an <insert statement> shall be a <value specification>.

b) A <row value constructor element> shall not specify DEFAULT.

19) Subclause 7.8, "<joined table>":

a) Conforming Entry SQL language shall not contain any <joined table>.

20) Subclause 7.9, "<where clause>":

a) A <value expression> directly contained in the <search condition> shall not include a reference to a column derived from a function.

21) Subclause 7.12, "<query specification>":

a) A <query specification> is not updatable if the <where clause> of the <table expression> contains a <subquery>.

b) A <select sublist> shall not specify ''<qualifier>. *''.

22) Subclause 7.13, "<query expression>":

a) A <query expression> shall not specify EXCEPT.

b) A <query term> shall not specify INTERSECT.

c) A <query primary> shall not contain a <joined table>.

d) A <query expression> shall not specify CORRESPONDING.

e) If UNION is specified, then except for column names, the descriptions of the first and second operands shall be identical and the description of the result is identical to the description of the operands.

23) Subclause 7.15, "<scalar subquery>, <row subquery>, and <table subquery>":

a) If a <subquery> is contained in a <comparison predicate>, then the <table expression> in the <query specification> shall not contain a <group by clause> or a <having clause> and shall not identify a grouped view.

b) The <query expression> contained in a <subquery> shall be a <query specification>.

24) Subclause 8.1, "<predicate>":

a) Conforming Entry SQL language shall not contain any <overlaps predicate>.

b) Conforming Entry SQL language shall not contain any <unique predicate>.

c) Conforming Entry SQL language shall not contain any <distinct predicate>.

25) Subclause 8.5, "<like predicate>":

a) The <match value> shall be a <column reference>.

b) A <pattern> shall be a <value specification>.

c) An <escape character> shall be a <value specification>.

26) Subclause 8.7, "<null predicate>":

a) A <row value constructor> shall be a <column reference>.

27) Subclause 8.10, "<unique predicate>":

a) Conforming Entry SQL language shall not contain any <unique predicate>.

28) Subclause 8.12, "<overlaps predicate>":

a) Conforming Entry SQL language shall not contain any <overlaps predicate>.

29) Subclause 8.15, "<distinct predicate>":

a) Conforming Entry SQL language shall not contain any <distinct predicate>.

30) Subclause 10.1, "<interval qualifier>":

a) Conforming Entry SQL language shall not contain any <interval qualifier>.

31) Subclause 10.2, "<language clause>":

a) Conforming Entry SQL language shall not contain a <language clause> that specifies MUMPS.

32) Subclause 10.6, "<character set specification>":

a) Conforming Entry SQL language shall contain no <character set specification>.

33) Subclause 10.8, "<constraint name definition> and <constraint attributes>":

a) Conforming Entry SQL language shall contain no <constraint name definition>.

34) Subclause 11.1, "<schema definition>":

a) A <schema element> shall not be a <domain definition>.

b) A <schema name clause> shall specify AUTHORIZATION and shall not specify a <schema name>.

c) A <schema character set specification> shall not be specified.

d) Conforming Intermediate SQL language shall not contain any <character set definition>.

35) Subclause 11.2, "<drop schema statement>":

a) Conforming Entry SQL language shall not contain a <drop schema statement>.

36) Subclause 11.6, "<table constraint definition>":

a) Conforming Entry SQL language shall contain no <constraint name definition>.

37) Subclause 11.21, "<drop table statement>":

a) Conforming Entry SQL language shall not contain any <drop table statement>.

38) Subclause 11.4, "<column definition>":

a) A <column definition> shall not contain a <domain name>.

b) A <column constraint> shall not contain a <referential triggered action>.

c) Conforming Entry SQL language shall not contain any <constraint name definition>.

39) Subclause 11.5, "<default clause>":

a) A <default option> shall not specify CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, SYSTEM_USER, SESSION_USER, or CURRENT_USER.

40) Subclause 11.6, "<table constraint definition>":

a) Conforming Entry SQL language shall not contain any <constraint name definition>.

41) Subclause 11.7, "<unique constraint definition>":

a) If PRIMARY KEY or UNIQUE is specified, then the <column definition> for each column whose <column name> is in the <unique column list> shall specify NOT NULL.

42) Subclause 11.8, "<referential constraint definition>":

a) A <referential constraint definition> shall not contain a <referential triggered action>.

43) Subclause 11.10, "<alter table statement>":

a) Conforming Entry SQL language shall not contain an <alter table statement>.

44) Subclause 11.11, "<add column definition>":

a) Conforming Entry SQL language shall not contain an <add column definition>.

45) Subclause 11.12, "<alter column definition>":

a) Conforming Entry SQL language shall not contain an <alter column definition>.

46) Subclause 11.13, "<set column default clause>":

a) Conforming Entry SQL language shall not contain a <set column default clause>.

47) Subclause 11.14, "<drop column default clause>":

a) Conforming Entry SQL language shall not contain a <drop column default clause>.

48) Subclause 11.16, "<drop column definition>":

a) Conforming Entry SQL language shall not contain a <drop column definition>.

49) Subclause 11.19, "<add table constraint definition>":

a) Conforming Entry SQL language shall not contain an <add table constraint definition>.

50) Subclause 11.20, "<drop table constraint definition>":

a) Conforming Entry SQL language shall not contain a <drop table constraint definition>.

51) Subclause 11.22, "<view definition>":

a) The <query expression> in a <view definition> shall be a <query specification>.

52) Subclause 11.23, "<drop view statement>":

a) Conforming Entry SQL language shall not contain a <drop view statement>.

53) Subclause 11.52, "<grant statement>":

a) An <object name> shall not specify TABLE.

b) In Conforming Entry SQL language, an <object name> shall not specify CHARACTER SET or DOMAIN.

54) Subclause 11.57, "<revoke statement>":

a) Conforming Entry SQL language shall not contain a <revoke statement>.

55) Subclause 11.24, "<domain definition>":

a) Conforming Entry SQL language shall not contain any <domain definition>.

56) Subclause 11.30, "<drop domain statement>":

a) Conforming Entry SQL language shall not contain a <drop domain statement>.

57) Subclause 11.33, "<character set definition>":

a) Conforming Entry SQL language shall not specify any <character set definition>.

58) Subclause 11.34, "<drop character set statement>":

a) Conforming Entry SQL language shall contain no <drop character set statement>.

59) Subclause 13.1, "<module>":

a) A <module> shall be associated with an SQL-agent during its execution. An SQL-agent shall be associated with at most one <module>.

b) A <module contents> shall not be a <dynamic declare cursor>.

c) A <module character set specification> shall not be specified.

60) Subclause 13.2, "<routine>":

a) A <parameter declaration> shall not specify a <data type> that is CHARACTER VARYING.

61) Subclause 13.3, "<SQL procedure statement>":

a) An <SQL procedure statement> shall not be an <SQL schema statement>.

b) An <SQL procedure statement> shall not be an <SQL dynamic statement>.

c) An <SQL session statement> shall not be a <set session authorization identifier statement> or a <set time zone statement>.

d) An <SQL transaction statement> shall not be a <set transaction statement>.

e) An <SQL procedure statement> shall not be an <SQL diagnostics statement>.

62) Subclause 15.1, "<declare cursor>":

a) A <declare cursor> shall not specify SCROLL.

b) A <cursor specification> shall not contain an <updatability clause>.

63) Subclause 15.3, "<fetch statement>":

a) A <fetch statement> shall not specify FROM.

b) If the data type of the target identified by the *i*-th <target specification> in the <fetch target list> is an exact numeric type, then the data type of the *i*-th column of the table *T* shall be an exact numeric type.

64) Subclause 15.5, "<select statement: single row>":

a) If the data type of the target identified by the *i*-th <target specification> in the <fetch target list> is an exact numeric type, then the data type of the *i*-th column of the table *T* shall be an exact numeric type.

65) Subclause 15.8, "<insert statement>":

a) The <query expression> that is contained in an <insert statement> shall be a <table value constructor> that specifies VALUES and contains exactly one <row value constructor> or it shall be a <query specification>.

b) If the data type of the target identified by the *i*-th <column name> is an exact numeric type, then the data type of the *i*-th item of the <insert statement> shall be an exact numeric type.

c) If the data type of the target *C* identified by the *i*-th <column name> is character string, then the length in characters of the *i*-th item of the <insert statement> shall be less than or equal to the length of *C*.

d) The <insert columns and source> shall immediately contain a <query expression>.

66) Subclause 15.9, "<update statement: positioned>":

a) If the data type of the column identified by the *i*-th <object column> is an exact numeric type, then the data type of the *i*-th <value expression> in the <update statement: positioned> shall be an exact numeric type.

b) If the data type of the column identified by the *i*-th <object column> *C* is character string, then the length in characters of the *i*-th <value expression> in the <update statement: positioned> shall be less than or equal to the length of *C*.

c) An <update source> shall not specify DEFAULT.

67) Subclause 15.10, "<update statement: searched>":

a) If the data type of the column identified by the *i*-th <object column> is an exact numeric type, then the data type of the *i*-th <value expression> in the <update statement: searched> shall be an exact numeric type.

b) If the data type of the column identified by the *i*-th <object column> *C* is character string, then the length in characters of the *i*-th <value expression> in the <update statement: searched> shall be less than or equal to the length of *C*.

68) Subclause 16.1, "<set transaction statement>":

a) Conforming Entry SQL language shall not contain any <set transaction statement>.

69) Subclause 16.6, "<commit statement>":

a) In conforming Entry SQL language, WORK shall be specified.

70) Subclause 16.7, "<rollback statement>":

a) In conforming Entry SQL language, WORK shall be specified.

71) Subclause 18.4, "<set session authorization identifier statement>":

a) Conforming Intermediate SQL language shall not contain any <set session authorization identifier statement>.

72) Subclause 18.6, "<set local time zone statement>":

a) Conforming Entry SQL language shall not contain any <set local time zone statement>.

73) Subclause 19.2, "<allocate descriptor statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

74) Subclause 19.3, "<deallocate descriptor statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

75) Subclause 19.4, "<get descriptor statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

76) Subclause 19.5, "<set descriptor statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

77) Subclause 19.6, "<prepare statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

78) Subclause 19.7, "<deallocate prepared statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

79) Subclause 19.8, "<describe statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

80) Subclause 19.9, "<using clause>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

81) Subclause 19.10, "<execute statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

82) Subclause 19.11, "<execute immediate statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

83) Subclause 19.12, "<dynamic declare cursor>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

84) Subclause 19.13, "<allocate cursor statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

85) Subclause 19.14, "<dynamic open statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

86) Subclause 19.15, "<dynamic fetch statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

87) Subclause 19.16, "<dynamic close statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

88) Subclause 19.17, "<dynamic delete statement: positioned>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

89) Subclause 19.18, "<dynamic update statement: positioned>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

90) Subclause 19.19, "<preparable dynamic delete statement: positioned>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

91) Subclause 19.20, "<preparable dynamic update statement: positioned>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

92) Subclause 20.1, "<get diagnostics statement>":

a) Conforming Entry SQL language shall not contain any <get diagnostics statement>.

93) Subclause 21.1, "<embedded SQL host program>":

a) An <embedded SQL statement> shall not contain a <dynamic declare cursor>.

b) An <embedded SQL statement> shall not contain an <embedded character set declaration>.

94) Subclause 21.3, "<embedded SQL Ada program>":

a) An <Ada qualified type specification> shall not specify a <character set specification>.

95) Subclause 21.4, "<embedded SQL C program>":

a) A <C derived variable> shall not be a <C VARCHAR variable> or a <C extended char variable>.

96) Subclause 21.5, "<embedded SQL COBOL program>":

a) A <COBOL character type> shall not specify a <character set specification>.

b) A <COBOL integer type> shall not be a <COBOL binary integer>.

97) Subclause 21.6, "<embedded SQL Fortran program>":

a) A <Fortran type specification> shall not specify a <character set specification>.

98) Subclause 21.8, "<embedded SQL Pascal program>":

a) A <Pascal type specification> shall not specify a <character set specification>.

99) Subclause 21.9, "<embedded SQL PL/I program>":

a) A <PL/I type specification> shall not specify CHARACTER VARYING.

b) A <PL/I type specification> shall not specify a <character set specification>.

100) Subclause 22.1, "<direct SQL statement>":

a) A <direct SQL statement> shall not be an <SQL schema statement>.

b) An <SQL session statement> shall not be a <set session authorization identifier statement> or a <set local time zone statement>.

c) A <direct SQL data statement> shall not be a <temporary table declaration>.

101) Subclause 23.2, "Information Schema":

a) Conforming Entry SQL language shall not reference any views from the Information Schema.

Annex B
(Informative)

Implementation-defined elements

This Annex references those features that are identified in the body of this
ANSI American
ISO International
Standard as implementation-defined.

The term *implementation-defined* is used to identify characteristics that may differ between implementations, but that shall be defined for each particular implementation.

1) Subclause 4.2.1, "Character strings and collating sequences": The specific character set associated with the subtype of character string represented by the <key word>s NATIONAL CHARACTER is implementation-defined.

2) Subclause 4.4, "Numbers": Whether truncation or rounding is performed when trailing digits are removed from a numeric value is implementation-defined.

3) Subclause 4.4, "Numbers": When an approximation is obtained by truncation or rounding and there are more than one approximation, then it is implementation-defined which approximation is chosen.

4) Subclause 4.4, "Numbers": It is implementation-defined which numerical values have approximations obtained by rounding or truncation for a given approximate numeric type.

5) Subclause 4.4, "Numbers": The boundaries within which the normal rules of arithmetic apply are implementation-defined.

6) Subclause 4.11, "Type conversions and mixing of data types": When converting between numeric data types, if least significant digits are lost, then it is implementation-defined whether rounding or truncation occurs.

7) Subclause 4.21, "Catalogs": The creation and destruction of catalogs is accomplished by implementation-defined means.

8) Subclause 4.21, "Catalogs": The set of catalogs that can be referenced in any SQL-statement, during any particular SQL-transaction, or during the course of an SQL-session is implementation-defined.

9) Subclause 4.21, "Catalogs": The default catalog for <module>s whose <module authorization clause> does not specify an explicit <catalog name> to qualify <schema name> and the default

catalog name substitution value for execution of <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s are implementation-defined.

10) Subclause 4.24, "SQL-environment": The constituents of an SQL-environment beyond those specified in Subclause 4.24, "SQL-environment", are implementation-defined.

11) Subclause 4.24, "SQL-environment": The rules determining whether a <module> is within the environment are implementation-defined.

12) Subclause 4.25, "Modules": The mechanisms by which <module>s are created or destroyed are implementation-defined.

13) Subclause 4.25, "Modules": The manner in which an association between a <module> and an SQL-agent is defined is implementation-defined.

14) Subclause 4.25, "Modules": Whether a compilation unit may invoke or transfer control to other compilation units, written in the same or a different programming language is implementation-defined.

15) Subclause 4.27.1, "Status parameters": The negative (exception) values for the SQLCODE status parameter are implementation-defined.

16) Subclause 4.32, "Embedded syntax": Whether a portion of the name space is reserved by an implementation for the names of procedures, subroutines, program variables, branch labels, <module>s, or <routine>s is implementation-defined; if a portion of the name space is so reserved, the portion reserved is also implementation-defined.

17) Subclause 4.33, "SQL dynamic statements": Within an SQL-session, all prepared statements belong to the same implementation-defined <module> that is different from any other <module> that exists simultaneously in the environment.

18) Subclause 4.37, "SQL-transactions": It is implementation-defined whether or not the non-dynamic or dynamic execution of an SQL-data statement or the execution of an <SQL dynamic data statement> is permitted to occur within the same SQL-transaction as the non-dynamic or dynamic execution of an SQL-schema statement. If it does occur, then the effect on any open cursor, prepared dynamic statement, or deferred constraint is also implementation-defined.

19) Subclause 4.37, "SQL-transactions": If an implementation detects unrecoverable errors and implicitly initiates the execution of a <rollback statement>, an exception condition is raised with an *implementation-defined exception code*.

20) Subclause 4.38, "SQL-connections": It is implementation-defined how an implementation uses <SQL-server name> to determine the location, identity, and communication protocol required to access the SQL-environment and create an SQL-session.

21) Subclause 4.39, "SQL-sessions": When an SQL-session is initiated other than through the use of an explicit <connect statement>, then an SQL-session associated with an implementation-defined SQL-environment is initiated. The default SQL-environment is implementiond-defined.

22) Subclause 4.39, "SQL-sessions": The mechanism and rules by which an environment determines whether a call to a <routine> is the last call within the last active <module> is implementation-defined.

23) Subclause 4.39, "SQL-sessions": An SQL-session uses one or more implementation-defined schemas that contain the instances of any global temporary tables, created local temporary tables, or declared local temporary tables within the SQL-session.

24) Subclause 4.39, "SQL-sessions": When an SQL-session is initiated, there is an implementation-defined default time zone used as the current default time zone displacement of the SQL-session.

25) Subclause 4.39, "SQL-sessions": When an SQL-session is initiated, there is an implementation-defined default catalog whose name is used to effectively qualify all unqualified <schema name>s contained in <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s.

26) Subclause 4.39, "SQL-sessions": When an SQL-session is initiated, there is an implementation-defined default schema whose name is used to effectively qualify all unqualified <schema qualified name>s contained in <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s.

27) Subclause 4.39, "SQL-sessions": When an SQL-session is initiated other than through the use of an explicit <connect statement>, there is an implementation-defined default <authorization identifier> that is used to for privilege checking for the execution of <SQL procedure statement>s contained in having an explicit <module authorization identifier>.

28) Subclause 4.34, "Direct invocation of SQL": The method of invoking <direct SQL statement>s, the method of raising conditions as a result of <direct SQL statement>s, the method of accessing diagnostic information, and the method of returning the results are all implementation-defined.

29) Subclause 5.1, "<SQL terminal character>": The end-of-line indicator (<newline>) is implementation-defined.

30) Subclause 5.3, "<literal>": The <character set name> character set used to represent national characters is implementation-defined.

31) Subclause 5.4, "Names and identifiers": If a <schema name> contained in a <schema name clause> but not contained in a <module> does not contain a <catalog name>, then an implementation-defined <catalog name> is implicit.

32) Subclause 5.4, "Names and identifiers": If a <schema name> contained in a <preparable statement> that is dynamically prepared in the current SQL-session through the execution of a <prepare statement> or an <execute immediate statement> does not contain a <catalog name>, then the implementation-defined <catalog name> for the SQL-session is implicit.

33) Subclause 5.4, "Names and identifiers": If a <schema name> contained in a <module authorization clause> does not contain a <catalog name>, then an implementation-defined <catalog name> is implicit.

34) Subclause 5.4, "Names and identifiers": If a <schema qualified name> contained in a <preparable statement> that is dynamically prepared in the current SQL-session through the execution of a <prepare statement> or an <execute immediate statement> does not contain a <schema name>, then the implementation-defined <schema name> for the SQL-session is implicit.

35) Subclause 5.4, "Names and identifiers": Those <identifier>s that are valid <authorization identifier>s are implementation-defined.

36) Subclause 5.4, "Names and identifiers": Those <identifier>s that are valid <catalog name>s are implementation-defined.

37) Subclause 5.4, "Names and identifiers": All <form-of-use conversion name>s are implementation-defined.

38) Subclause 5.4, "Names and identifiers": The <entry name> of an entry point to a function defined as part of an abstract data type is implementation-defined.

39) Subclause 6.1, "<data type>": The <character set name> associated with NATIONAL CHARACTER is implementation-defined.

40) Subclause 6.1, "<data type>": If a <precision> is omitted, then an implementation-defined <precision> is implicit.

41) Subclause 6.1, "<data type>": The decimal precision of a data type defined as DECIMAL for each value specified by <precision> is implementation-defined.

42) Subclause 6.1, "<data type>": The precision of a data type defined as INTEGER is implementation-defined, but has the same radix as that for SMALLINT.

43) Subclause 6.1, "<data type>": The precision of a data type defined as SMALLINT is implementation-defined, but has the same radix as that for INTEGER.

44) Subclause 6.1, "<data type>": The binary precision of a data type defined as FLOAT for each value specified by <precision> is implementation-defined.

45) Subclause 6.1, "<data type>": The precision of a data type defined as REAL is implementation-defined.

46) Subclause 6.1, "<data type>": The precision of a data type defined as DOUBLE PRECISION is implementation-defined, but greater than that for REAL.

47) Subclause 6.1, "<data type>": For every <data type>, the limits of the <data type> are implementation-defined.

48) Subclause 6.1, "<data type>": The maximum lengths for character string types, variable-length character string types, bit string types, and variable-length bit string types are implementation-defined.

49) Subclause 6.1, "<data type>": If CHARACTER SET is not specified for <character string type>, then the character set is implementation-defined.

50) Subclause 6.1, "<data type>": The character set named SQL_TEXT is an implementation-defined character set that contains every character that is in <SQL language character> and all characters that are in other character sets supported by the implementation.

51) Subclause 6.1, "<data type>": For the <exact numeric type>s DECIMAL and NUMERIC, the maximum values of <precision> and of <scale> are implementation-defined.

52) Subclause 6.1, "<data type>": For the <approximate numeric type> FLOAT, the maximum value of <precision> is implementation-defined.

53) Subclause 6.1, "<data type>": For the <approximate numeric type>s FLOAT, REAL, and DOUBLE PRECISION, the maximum and minimum values of the exponent are implementation-defined.

54) Subclause 6.1, "<data type>": The maximum value of <time fractional seconds precision> is implementation-defined, but shall not be less than 6.

55) Subclause 6.1, "<data type>": The maximum values of <time precision> and <timestamp precision> for a <datetime type> are the same implementation-defined value.

56) Subclause 6.2, "<value specification> and <target specification>": Whether the character string of the <value specification>s CURRENT_USER, SESSION_USER, and SYSTEM_USER is variable-length or fixed-length, and its maximum lenth if it is variable-length or its length if it is fixed-length, are implementation-defined.

57) Subclause 6.2, "<value specification> and <target specification>": The value specified by SYSTEM_USER is an implementation-defined string that represents the operating system user who executed the <module> that contains the SQL-statement whose execution caused the SYSTEM_USER <general value specification> to be evaluated.

58) Subclause 6.2, "<value specification> and <target specification>": In Intermediate SQL, the specific data type of <indicator parameter>s and <indicator variable>s shall be the same implementation-defined data type.

ISO Only–SQL3

59) Subclause 6.2, "<value specification> and <target specification>": The length of the fixed-length character string of the <value specification> USER is implementation-defined.

60) Subclause 6.7, "<set function specification>": The precision of the value derived from application of the COUNT function is implementation-defined.

61) Subclause 6.7, "<set function specification>": The precision of the value derived from application of the SUM function to a data type of exact numeric is implementation-defined.

62) Subclause 6.7, "<set function specification>": The precision and scale of the value derived from application of the AVG function to a data type of exact numeric is implementation-defined.

63) Subclause 6.7, "<set function specification>": The precision of the value derived from application of the SUM function or AVG function to a data type of approximate numeric is implementation-defined.

64) Subclause 6.8, "<numeric value function>": The precision of <position expression> is implementation-defined.

65) Subclause 6.8, "<numeric value function>": The precision of <extract expression> is implementation-defined. If <datetime field> specifies SECOND, then the scale is also implementation-defined.

66) Subclause 6.8, "<numeric value function>": The precision of <length expression> is implementation-defined.

67) Subclause 6.9, "<string value function>": The maximum length of <character translation> or <form-of-use conversion> is implementation-defined.

68) Subclause 6.13, "<cast specification>": Whether to round or truncate when casting to exact or approximate numeric data types is implementation-defined.

69) Subclause 6.15, "<numeric value expression>": When the data type of both operands of the addition. subtraction, multiplication, or division operator is exact numeric, the precision of the result is implementation-defined.

70) Subclause 6.15, "<numeric value expression>": When the data type of both operands of the division operator is exact numeric, the scale of the result is implementation-defined.

71) Subclause 6.15, "<numeric value expression>": When the data type of either operand of an arithmetic operator is approximate numeric, the precision of the result is implementation-defined.

72) Subclause 6.15, "<numeric value expression>": Whether to round or truncate when performing division is implementation-defined.

73) Subclause 6.19, "<interval value expression>": When an interval is produced from the difference of two datetimes, the choice of whether to round or truncate is implementation-defined.

74) Subclause 7.14, "<recursive union>": The implicit limit value in a <limit clause> is an implementation-defined positive integer.

75) Subclause 7.14, "<recursive union>": The order of rows returned when PREORDER is specified is implementation-defined.

76) Subclause 9.1, "Retrieval assignment": If a value *V* is approximate numeric and a target *T* is exact numeric, then whether the approximation of *V* retrieved into *T* is obtained by rounding or by truncation is implementation-defined.

77) Subclause 9.2, "Store assignment": If a value *V* is approximate numeric and a target *T* is exact numeric, then whether the approximation of *V* stored into *T* is obtained by rounding or by truncation is implementation-defined.

78) Subclause 9.3, "Set operation result data types and nullabilities": If all of the data types in *DTS* are exact numeric, then the result data type is exact numeric with implementation-defined precision.

79) Subclause 9.3, "Set operation result data types and nullabilities": If any data type in *DTS* is approximate numeric, then each data type in *DTS* shall be numeric and the result data type is approximate numeric with implementation-defined precision.

80) Subclause 10.1, "<interval qualifier>": The maximum value of <interval leading field precision> is implementation-defined, but shall not be less than 2.

81) Subclause 10.1, "<interval qualifier>": The maximum value of <interval fractional seconds precision> is implementation-defined, but shall not be less than 2.

82) Subclause 10.3, "<routine invocation>": Whether or not *SAI* can invoke <routine>s in a <module> with explicit <module authorization identifier> *MAI* is implementation-defined, as are any restrictions pertaining to such invocation.

83) Subclause 10.3, "<routine invocation>": The method and time of binding of the designated external routine to the schema or <module> containing the <routine> is implementation-defined.

84) Subclause 10.6, "<character set specification>": The <standard character repertoire name>s, <implementation-defined character repertoire name>s, <standard universal character form-of-use name>s, and <implementation-defined universal character form-of-use name>s that are supported are implementation-defined.

85) Subclause 11.1, "<schema definition>": If <schema character set specification> is not specified, then a <schema character set specification> containing an implementation-defined <character set specification> is implicit.

86) Subclause 11.1, "<schema definition>": If AUTHORIZATION <authorization identifier> is not specified, then an <authorization identifier> equal to the implementation-defined <authorization identifier> for the SQL-session is implicit.

87) Subclause 11.1, "<schema definition>": The privileges necessary to execute the <schema definition> are implementation-defined.

88) Subclause 11.53, "<role definition>": The Access Rules are implementation-defined.

89) Subclause 11.35, "<collation definition>": The <standard collation name>s and <implementation-defined collation name>s that are supported are implementation-defined.

90) Subclause 11.35, "<collation definition>": The collating sequence resulting from the specification of EXTERNAL in a <collation definition> may be implementation-defined.

91) Subclause 11.37, "<translation definition>": The <standard translation name>s and <implementation-defined translation name>s that are supported are implementation-defined.

92) Subclause 11.57, "<revoke statement>": When loss of the USAGE privilege on a character set cuases a module to be determined to be a lost module, the impact on that module is implementation-defined.

93) Subclause 13.1, "<module>": If the explicit or implicit <schema name> does not specify a <catalog name>, then an implementation-defined <catalog name> is implicit.

94) Subclause 13.1, "<module>": If a <module character set specification> is not specified, then a <module character set specification> that specifies the implementation-defined character set that contains every character that is in <SQL language character> is implicit.

95) Subclause 13.1, "<module>": If a <module> that contains a <routine> is associated with an SQL-agent that is associated with another <module> that contains a <routine> with the same <procedure name>, then the effect is implementation-defined.

96) Subclause 13.2, "<routine>": There is an implementation-defined package and character type for use in Ada bindings. The precisions and scales of the Ada types for SQL data types and the <exact numeric type> of indicator parameters are implementation-defined.

97) Subclause 13.2, "<routine>": The number of bits in a C character is implementation-defined.

98) Subclause 13.2, "<routine>": The precision of an SQLCODE parameter in an <embedded COBOL program> is an implementation-defined value between 4 and 18, inclusive.

99) Subclause 13.2, "<routine>": The number of bits contained in a COBOL character is implementation-defined.

100) Subclause 13.2, "<routine>": The precision of a COBOL data type corresponding to SQL INTEGER or SMALLINT is implementation-defined.

101) Subclause 13.2, "<routine>": The number of bits contained in a Fortran character is implementation-defined.

102) Subclause 13.2, "<routine>": The maximum length of the MUMPS variable-length character type is implementation-defined.

103) Subclause 13.2, "<routine>": The number of bits contained in a Pascal character is implementation-defined.

104) Subclause 13.2, "<routine>": The precision of an SQLCODE parameter in an <embedded PL/I program> is implementation-defined.

105) Subclause 13.2, "<routine>": The precision of a PL/I data type corresponding to SQL INTEGER or SMALLINT is implementation-defined.

106) Subclause 13.2, "<routine>": If the value of any input parameter provided by the SQL-agent falls outside the set of allowed values of the data type of the parameter, or if the value of any output parameter resulting from the execution of the <routine> falls outside the set of values supported by the SQL-agent for that parameter, then the effect is implementation-defined.

107) Subclause 13.2, "<routine>": If the <module> that contains the <routine> is associated with an SQL-agent that is associated with another <module> that contains a <routine> with the same <procedure name>, then the effect is implementation-defined.

108) Subclause 13.2, "<routine>": If no session is active when the first <routine> in an SQL-session is called, and if the <SQL procedure statement> of the <routine> is not a <connect statement>, then an implementation-defined <connect statement> is effectively executed to create an SQL-session and connect the user to a default SQL-environment.

109) Subclause 13.2, "<routine>": The number of permitted outstanding asynchronous SQL-statements is implementation-defined.

110) Subclause 13.2, "<routine>": The null character that defines the end of a C character string is implementation-defined.

ANSI Only–SQL3

111) Subclause 13.2, "<routine>": If the <routine> is an external routine, then the method and time of binding of that external routine to the schema or <module> containing the <routine> is implementation-defined.

112) Subclause 15.1, "<declare cursor>": Whether null values shall be considered greater than or less than all non-null values in determining the order of rows in a table associated with a <declare cursor> is implementation-defined.

113) Subclause 15.1, "<declare cursor>": Whether an implementation is able to disallow significant changes that would not be visible through a currently open cursor is implementation-defined.

114) Subclause 15.2, "<open statement>": The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

115) Subclause 15.6, "<delete statement: positioned>": The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

116) Subclause 15.7, "<delete statement: searched>": The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

117) Subclause 15.8, "<insert statement>": The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

118) Subclause 15.9, "<update statement: positioned>": The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

119) Subclause 15.10, "<update statement: searched>": The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

120) Subclause 16.1, "<set transaction statement>": The isolation level that is set for a transaction is an implementation-defined isolation level that will not exhibit any of the phenomena that the explicit or implicit <level of isolation> would not exhibit, as specified in Table 9, "SQL-transaction isolation levels and the three phenomena".

121) Subclause 16.4, "<savepoint statement>": The maximum number of savepoints per SQL-transaction is implementation-defined.

122) Subclause 16.7, "<rollback statement>": The status of any open cursors in any <module> associated with the current SQL-transaction that were opened by that SQL-transaction before the establishment of a savepoint to which a rollback is executed is implementation-defined.

123) Subclause 17.1, "<connect statement>": If <user name> is not specified, then an implementation-defined <user name> for the SQL-connection is implicit.

124) Subclause 17.1, "<connect statement>": The restrictions on whether or not the <user name> must be identical to the <module authorization identifier> for the <module> that contains the <routine> that contains the <connect statement> are implementation-defined.

125) Subclause 17.1, "<connect statement>": If DEFAULT is specified, then the method by which the default SQL-environment is determined is implementation-defined.

126) Subclause 17.1, "<connect statement>": The method by which <SQL-server name> is used to determine the appropriate SQL-environment is implementation-defined.

127) Subclause 18.4, "<set session authorization identifier statement>": Whether or not the <authorization identifier> for the SQL-session can be set to an <authorization identifier> other than the <authorization identifier> of the SQL-session when the SQL-session is started is implementation-defined, as are any restrictions pertaining to such changes.

128) Subclause 19.2, "<allocate descriptor statement>": If WITH MAX <occurrences> is not specified, then an implementation-defined default value for <occurrences> that is greater than 0 is implicit.

129) Subclause 19.2, "<allocate descriptor statement>": The maximum number of SQL descriptor areas and the maximum number of item descriptor areas for a single SQL descriptor area are implementation-defined.

130) Subclause 19.5, "<set descriptor statement>": Restrictions on changing TYPE, LENGTH, OCTET_LENGTH, SCALE, COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME values resulting from the execution of a <describe statement> before execution of a <execute statement>, <dynamic open statement>, or <dynamic fetch statement> are implementation-defined, except as provided in the General Rules of Subclause 19.9, "<using clause>".

131) Subclause 19.6, "<prepare statement>": The Format and Syntax Rules for a <preparable implementation-defined statement> are implementation-defined.

132) Subclause 19.9, "<using clause>": The character set of the data type of <descriptor name> is implementation-defined.

133) Subclause 20.1, "<get diagnostics statement>": The actual length of variable-length character items in the diagnostics area is implementation-defined but not less than 128.

134) Subclause 20.1, "<get diagnostics statement>": The character string value set for CLASS_ORIGIN and SUBCLASS_ORIGIN for an implementation-defined class code or subclass code is implementation-defined, but shall not be 'ISO 9075'.

135) Subclause 20.1, "<get diagnostics statement>": The value of MESSAGE_TEXT is an implementation-defined character string.

136) Subclause 21.1, "<embedded SQL host program>": If an <embedded character set declaration> is not specified, then an <embedded character set declaration> containing an implementation-defined <character set specification> is implicit.

137) Subclause 21.1, "<embedded SQL host program>": Each <allocate cursor statement> is replaced with a host language procedure or subroutine call of an implementation-defined procedure that associates the <dynamic cursor name> with the prepared statement.

138) Subclause 21.3, "<embedded SQL Ada program>": SQLCODE_TYPE describes an exact numeric variable whose precision is the implementation-defined precision defined for the SQLCODE parameter.

139) Subclause 21.5, "<embedded SQL COBOL program>": The COBOL data description clauses, in addition to the PICTURE, SIGN, USAGE and VALUE clauses, that may appear in a <COBOL variable definition> are implementation-defined.

140) Subclause 21.5, "<embedded SQL COBOL program>": The precision of the <COBOL type specification> that corresponds to SQLCODE is implementation-defined.

141) Subclause 21.9, "<embedded SQL PL/I program>": The PL/I data description clauses, in addition to the <PL/I type specification> and the INITIAL clause, that may appear in a <PL/I variable definition> are implementation-defined.

142) Subclause 21.9, "<embedded SQL PL/I program>": The precision of the <PL/I type specification> that corresponds to SQLCODE is implementation-defined.

143) Subclause 22.1, "<direct SQL statement>": The <value specification> that represents the null value is implementation-defined.

144) Subclause 22.1, "<direct SQL statement>": The Format, Syntax Rules, and Access Rules for <direct implementation-defined statement> are implementation-defined.

145) Subclause 22.1, "<direct SQL statement>": Whether a <direct implementation-defined statement> may be associated with an active transaction is implementation-defined.

146) Subclause 22.1, "<direct SQL statement>": Whether a <direct implementation-defined statement> initiates a transaction is implementation-defined.

147) Subclause 23.3.15, "TABLE_CONSTRAINTS base table": If the containing <add table constraint definition> does not specify a <constraint name definition>, then the value of CONSTRAINT_NAME is implementation-defined.

148) Subclause 23.3.30, "CHARACTER_SETS base table": The values of FORM_OF_USE and NUMBER_OF_CHARACTERS, in the row for the character set INFORMATION_SCHEMA.SQL_TEXT, are implementation-defined.

149) Subclause 23.3.31, "COLLATIONS base table": The value of COLLATION_DEFINITION is implementation-defined.

150) Subclause 23.3.37, "SQL_LANGUAGES base table": The value of SQL_LANGUAGE_IMPLEMENTATION is implementation-defined. If the value of SQL_LANGUAGE_SOURCE is not 'ISO 9075', then the value of all other columns is implementation-defined.

151) Subclause 24.1, "SQLSTATE": The character set associated with the class value and subclass value of the SQLSTATE parameter is implementation-defined.

152) Subclause 24.1, "SQLSTATE": The values and meanings for classes and subclasses that begin with one of the <digit>s '5', '6', '7', '8', or '9' or one of the <simple Latin upper case letter>s 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', or 'Z' are implementation-defined. The values and meanings for all subclasses that are associated with implementation-defined class values are implementation-defined.

153) Subclause 24.2, "SQLCODE": The negative values returned in an SQLCODE parameter to indicate exception conditions are implementation-defined.

154) Clause 25, "Conformance": The method of flagging nonconforming SQL language or processing of conforming SQL language is implementation-defined, as is the list of additional <key word>s that may be required by the implementation.

Annex C
(Informative)

Implementation-dependent elements

This Annex references those places where this
ANSI American
ISO International
Standard states explicitly that the actions of a conforming implementation are implementation-dependent.

The term *implementation-dependent* is used to identify characteristics that may differ between implementations, but that are not necessarily specified for any particular implementation.

1) Subclause 3.3.4, "Use of terms": The effect on <target specification>s and SQL descriptor areas of an SQL-statement that terminates with an exception condition, unless explicitly defined by this
ANSI American
ISO International
standard, is implementation-dependent.

2) Subclause 3.3.4, "Use of terms": If more than one condition could have occurred as a result of execution of a statement, then it is implementation-dependent whether diagnostic information pertaining to more than one condition is made available.

3) Subclause 3.3.4, "Use of terms": The treatment of language that does not conform to the SQL Formats and Syntax Rules is implementation-dependent.

4) Subclause 3.3.4, "Use of terms": It is implementation-dependent whether expressions are actually evaluated left-to-right when the precedence is not otherwise determined by the Formats or by parentheses.

5) Subclause 3.3.4, "Use of terms": If evaluation of the inessential parts of an expression or search condition would cause an exception condition to be raised, then it is implementation-dependent whether or not that condition is raised.

6) Subclause 3.3.3, "Specification of the Information Schema" The actual objects on which the Information Schema views are based are implementation-dependent.

7) Subclause 4.1, "Data types": The physical representation of a value of a data type is implementation-dependent.

8) Subclause 4.1, "Data types": The null value or values for each data type is implementation-dependent.

9) Subclause 4.13, "Nulls": A null state is a classified null value that is implementation-dependent but distinct from the both the general null value and all other classified null values for the domain or column.

10) Subclause 4.16, "Tables": Because global temporary table contents are distinct within SQL-sessions, and created local temporary tables are distinct within <module>s within SQL-sessions, the effective <schema name> of the schema in which the global temporary table or the created local temporary table is instantiated is an implementation-dependent <schema name> that may be thought of as having been effectively derived from the <schema name> of the schema in which the global temporary table or created local temporary table is defined and the implementation-dependent SQL-session identifier associated with the SQL-session.

11) Subclause 4.16, "Tables": The effective <schema name> of the schema in which the created local temporary table is instantiated may be thought of as being further qualified by a unique implementation-dependent name associated with the <module> in which the created local temporary table is referenced.

12) Subclause 4.16, "Tables": Whether a temporary viewed table is materialized is implementation-dependent.

13) Subclause 4.28, "Diagnostics area": The actual size of the diagnostics area is implementation-dependent when the SQL-agent does not specify the size.

14) Subclause 4.28, "Diagnostics area": The ordering of the the information about conditions placed into the diagnostics area is implementation-dependent, except that the first condition in the diagnostics area always corresponds to the condition corresponding to the SQLSTATE or SQLCODE value.

15) Subclause 4.30, "Cursors": If the <declare cursor> does not include an <order by clause>, or includes an <order by clause> that does not specify the order of the rows completely, then the rows of the table have an order that is defined only to the extent that the <order by clause> specifies an order and is otherwise implementation-dependent.

16) Subclause 4.30, "Cursors": When the ordering of a cursor is not defined by an <order by clause>, the relative position of two rows is implementation-dependent.

17) Subclause 4.30, "Cursors": The effect on the position and state of an open cursor when an error occurs during the execution of an SQL-statement that identifies the cursor is implementation-dependent.

18) Subclause 4.30, "Cursors": If a cursor is open and a change is made to SQL-data from within the same SQL-transaction other than through that cursor, and the cursor was not declared SENSITIVE or INSENSITIVE, then whether that change will be visible through that cursor before it is closed is implementation-dependent.

19) Subclause 4.35, "Privileges and roles": The mapping of <authorization identifier>s to operating system users is implementation-dependent.

20) Subclause 4.35, "Privileges and roles": When an SQL-session is initiated, the current <authorization identifier> for the SQL-session is determined in an implementation-dependent manner, unless the session is initiated using a <connect statement>.

21) Subclause 4.36, "SQL-agents": An SQL-agent is an implementation-dependent entity that causes the execution of SQL-statements.

22) Subclause 4.37, "SQL-transactions": The schema definitions that are implicitly read on behalf of executing an SQL-statement are implementation-dependent.

23) Subclause 4.39, "SQL-sessions": A unique implementation-dependent SQL-session identifier is associated with each SQL-session.

24) Subclause 4.40, "Client-server operation": The <module name> of the <module> that is effectively materialized on an SQL-server is implementation-dependent.

25) Subclause 4.40, "Client-server operation": Diagnostic information is passed to the diagnostics area in the SQL-client is passed in an implementation-dependent manner.

26) Subclause 4.40, "Client-server operation": The effect on diagnostic information of incompatibilities between the character repertoires supported by the SQL-client and SQL-server environments is implementation-dependent.

ANSI Only–SQL3

27) Subclause 5.3, "<literal>": The method of representing an object identifier as an <oid value> is implementation-dependent.

28) Subclause 5.3, "<literal>": Within the definition of an <oid literal>, the allowed <oid value>s are implementation-dependent.

29) Subclause 6.6, "<column reference> and <object reference>": The implicit qualifier of a <column reference> for which there is more than one possible qualifier with most local scope is implementation-dependent.

30) Subclause 6.10, "<datetime value function>": The time of evaluation of the CURRENT_DATE, CURRENT_TIME, and CURRENT_TIMESTAMP functions during the execution of an SQL-statement is implementation-dependent.

31) Subclause 6.19, "<interval value expression>": The start datetime used for converting intervals to scalars for subtraction purposes is implementation-dependent.

32) Subclause 7.1, "<row value constructor>": The names of the columns of a <row value constructor> that specifies a <row value constructor list> are implementation-dependent.

33) Subclause 7.12, "<query specification>": When a column is not named by an <as clause> and is not derived from a single <column reference>, then the name of the column is implementation-dependent.

34) Subclause 7.12, "<query specification>": If a <simple table> is neither a <query specification> nor an <explicit table>, then the name of each column of the <simple table> is implementation-dependent.

35) Subclause 7.12, "<query specification>": If a <non-join query term> is not a <non-join query primary> and the <column name> of the corresponding columns of both tables participating in the <non-join query term> are not the same, then the result column has an implementation-dependent <column name>.

36) Subclause 7.12, "<query specification>": If a <non-join query expression> is not a <non-join query term> and the <column name> of the corresponding columns of both tables participating in the <non-join query expression > are not the same, then the result column has an implementation-dependent <column name>.

37) Subclause 8.2, "<comparison predicate>": When the operations MAX, MIN, DISTINCT, and references to a grouping column refer to a variable-length character string or a variable-length bit string, the specific value selected from the set of equal values is implementation-dependent.

38) Subclause 9.3, "Set operation result data types and nullabilities": The specific character set chosen for the result is implementation-dependent, but shall be the character set of one of the data types in *DTS*.

39) Subclause 10.3, "<routine invocation>": The result data item supplied has an implementation-dependent value.

40) Subclause 10.3, "<routine invocation>": If a parameter has an indicator parameter that specifies the parameter to be null, then the value of the parameter itself is implementation-dependent.

41) Subclause 10.3, "<routine invocation>": If the external function is not a standard-conforming program, then the effect of the <external function invocation> is implementation-dependent.

42) Subclause 11.3, "<table definition>": The <abstract data type name> of an abstract data type specified in a <table definition> without specifying ''OF NEW TYPE *ADTN*'' is implementation-dependent.

43) Subclause 11.6, "<table constraint definition>": The <constraint name> of a constraint that does not specify a <constraint name definition> is implementation-dependent.

44) Subclause 11.8, "<referential constraint definition>": The specific value to use for cascading among various values that are not distinct is implementation-dependent.

45) Subclause 11.24, "<domain definition>": The <constraint name> of a constraint that does not specify a <constraint name definition> is implementation-dependent.

46) Subclause 11.35, "<collation definition>": The collation of characters for which a collation is not otherwise specified is implementation-dependent.

47) Subclause 13.1, "<module>": If the SQL-agent that performs a call of a <routine> in a <module> is not a standard program in the language specified in the <language clause> of the <module>, then the results are implementation-dependent.

48) Subclause 13.1, "<module>": If the <language clause> of a <module> specifies ADA and the SQL-agent performs calls of <routine>s in that <module> from more than one Ada task, then the results are implementation-dependent.

49) Subclause 13.2, "<routine>": After the execution of the last <routine>, if an unrecoverable error has not occurred, and the SQL-agent did not terminate unexpectedly, and no constraint is not satisfied, then the choice of whether to perform a <commit statement> or a <rollback statement> is implementation-dependent. The determination of whether an SQL-agent has terminated unexpectedly is implementation-dependent.

50) Subclause 13.2, "<routine>": If there are more than one status parameter, then the order in which values are assigned to these status parameters is implementation-dependent.

ANSI Only–SQL3

51) Subclause 13.2, "<routine>": If the designated external routine is not a program that conforms to the programming language standard specified by the <external routine language clause>, then the results are implementation-dependent.

52) Subclause 14.3, "<compound statement>": The variables, cursors, routines, and temporary tables and view specified in a <local declaration list> are created in an implementation-dependent order.

53) Subclause 14.6, "<SQL variable declaration>": When a variable associated with an <SQL variable declaration> is created and the <SQL variable declaration> does not contain a <default clause>, then the intial value of the variable is implementation-dependent.

54) Subclause 14.14, "<for statement": The <cursor name> used in the transformation of a <for statement> into a <loop statement> is implementation-dependent, as is the <variable name> used in the <loop statement> for getting diagnostics information.

55) Subclause 15.1, "<declare cursor>": If a <declare cursor> does not contain an
ANSI <order by clause> and the table is not an ordered table,
ISO <order by clause>,
then the ordering of rows in the table associated with that <declare cursor> is implementation-dependent.

56) Subclause 15.1, "<declare cursor>": If a <declare cursor> contains an <order by clause> and a group of two or more rows in the table associated with that <declare cursor> contain values that

Case:

a) are the same null value, or

b) compare equal according to Subclause 8.2, "<comparison predicate>" in all columns specified in the <order by clause>, then the order in which rows in that group are returned is implementation-dependent.

57) Subclause 15.3, "<fetch statement>": The order of assignment to targets in the <fetch target list> of values returned by a <fetch statement>, other than status parameters, is implementation-dependent.

58) Subclause 15.3, "<fetch statement>": If an error occurs during assignment of a value to a target during the execution of a <select statement: single row>, then the values of targets other than status parameters are implementation-dependent.

59) Subclause 15.5, "<select statement: single row>": The order of assignment to targets in the <select target list> of values returned by a <select statement: single row>, other than status parameters, is implementation-dependent.

60) Subclause 15.5, "<select statement: single row>": If the cardinality of the <query expression> is greater than 1, then it is implementation-dependent whether or not values are assigned to the targets identified by the <select target list>.

61) Subclause 15.5, "<select statement: single row>": If an error occurs during assignment of a value to a target during the execution of a <select statement: single row>, then the values of targets other than status parameters are implementation-dependent.

62) Subclause 15.12, "<temporary view definition>": Whether a temporary viewed table is materialized is implementation-dependent.

63) Subclause 16.1, "<set transaction statement>": If <number of conditions> is not specified, then an implementation-dependent value not less than 1 is implicit.

64) Subclause 16.4, "<savepoint statement>": If <savepoint specifier> is specified as <simple target specification>, then *S* is set to an implementation-dependent value that is non-0 and different from all other values that have been used to identify savepoints within the current SQL-transaction.

65) Subclause 17.3, "<disconnect statement>": If ALL is specified, then *L* is a list representing every active SQL-connection that has been established by a <connect statement> by the current SQL-agent and that has not yet been disconnected by a <disconnect statement>, in an implementation-dependent order.

66) Subclause 19.4, "<get descriptor statement>": If an exception condition is raised in a <get descriptor statement>, then the values of all targets specified by <simple target specification 1>, <simple target specification 2>, and <simple target specification 3> are implementation-dependent.

67) Subclause 19.5, "<set descriptor statement>": If an exception condition is raised in a <set descriptor statement>, then the values of all elements of the descriptor specified in the <set descriptor statement> are implementation-dependent.

68) Subclause 19.6, "<prepare statement>": The validity of an <extended statement name> value or a <statement name> in an SQL-transaction different from the one in which the statement was prepared is implementation-dependent.

69) Subclause 19.9, "<using clause>": When a <describe output statement> is executed, the values of DATA and INDICATOR are implementation-dependent. If TYPE indicates a character string type or a bit string type, then the values of SCALE and PRECISION are implementation-dependent. If TYPE indicates an exact or approximate numeric type, then the values of LENGTH and OCTET_LENGTh are implementation-dependent. If TYPE indicates a boolean type, then the values of PRECISION, SCALE, LENGTH, and OCTET_LENGTH are implementation-dependent.

70) Subclause 19.9, "<using clause>": When a <describe input statement> is used, the values for NAME, DATA, and INDICATOR in the SQL dynamic descriptor area structure is implementation-dependent. If TYPE indicates a character string type or a bit string type, then the values of SCALE and PRECISION are implementation-dependent. If TYPE indicates an exact or approximate numeric type, then the values of LENGTH and OCTET_LENGTh are implementation-dependent. If TYPE indicates a boolean type, then the values of PRECISION, SCALE, LENGTH, and OCTET_LENGTH are implementation-dependent.

71) Subclause 20.1, "<get diagnostics statement>": The value of ROW_COUNT following the execution of an SQL-statement that does not directly result in the execution of a <delete statement: searched>, an <insert statement>, or an <update statement: searched> is implementation-dependent.

72) Subclause 20.1, "<get diagnostics statement>": If <condition number> has a value other than 1, then the association between <condition number> values and specific conditions raised during evaluation of the General Rules for that SQL-statement is implementation-dependent.

73) Subclause 21.1, "<embedded SQL host program>": The <module name> of the implied <module> derived from an <embedded SQL host program> is implementation-dependent.

74) Subclause 21.1, "<embedded SQL host program>": The <module authorization identifier> of the implied <module> derived from an <embedded SQL host program> is implementation-dependent.

75) Subclause 21.1, "<embedded SQL host program>": In each <declare cursor> in the implied <module> derived from an <embedded SQL host program>, each <embedded variable name> has been replaced consistently with a distinct
ANSI <host parameter name>
ISO <parameter name>
that is implementation-dependent.

76) Subclause 21.1, "<embedded SQL host program>": The <procedure name> of each <routine> in the implied <module> derived from an <embedded SQL host program> is implementation-dependent.

77) Subclause 21.1, "<embedded SQL host program>": In each <routine> in the implied <module> derived from an <embedded SQL host program>, each <embedded variable name> has been replaced consistently with a distinct
ANSI <host parameter name>
ISO <parameter name>
that is implementation-dependent.

78) Subclause 21.1, "<embedded SQL host program>": For <SQL procedure statement>s other than <open statement>s, whether one <routine> in the implied <module> derived from an <embedded SQL host program> can correspond to more than one <SQL procedure statement> in the <embedded SQL host program> is implementation-dependent.

79) Subclause 21.1, "<embedded SQL host program>": In each <routine> in the implied <module> derived from an <embedded SQL host program>, the order of the instances of <parameter declaration> is implementation-dependent.

80) Subclause 22.1, "<direct SQL statement>": If an unrecoverable error has occurred, or if the direct invocation of SQL terminated unexpectedly, or if any constraint is not satisfied, then a <rollback statement> is performed. Otherwise, the choice of which of these SQL-statements to perform is implementation-dependent. The determination of whether a direct invocation of SQL has terminated unexpectedly is implementation-dependent.

81) Subclause 23.2, "Information Schema": The actual <query specification>s required to create the Information Schema tables are implementation-dependent.

82) Subclause 23.3.30, "CHARACTER_SETS base table": The value of DEFAULT_COLLATE_NAME for default collations specifying the order of characters in a repertoire is implementation-dependent.

Annex D
(Informative)

Deprecated features

It is intended that the following features will be removed at a later date from a revised version of this
ANSI American
ISO International
Standard:

1) SQLCODE

2) A <COBOL computational integer>s in <embedded SQL COBOL program>s.

3) The <unsigned integer> option of <sort specification> of <declare cursor>.

4) The <parameter declaration>s in a <routine> without enclosing parentheses and without commas separating multiple <parameter declaration>s.

5) The use of SQLERROR as a <condition> in an <embedded exception declaration>.

ISO Only–SQL3

6) The use of the <key word> USER.

Annex E
(Informative)

Incompatibilities with X3.135-1992 and ISO/IEC 9075:1992

This
[ANSI] American
[ISO] International
Standard introduces some incompatibilities with the earlier version of Database Language SQL as specified in
[ANSI] ANS X3.135-1992.
[ISO] ISO/IEC 9075:1992.
Unless specified in this Annex, features and capabilities of Database Language SQL are compatible with the earlier version of this
[ANSI] American
[ISO] International
Standard.

1) A number of additional <reserved word>s have been added to the language. These <reserved word>s are:

   — ACTION

   — ACTOR

   — AFTER

   — ALIAS

   — ASYNC

   — BEFORE

   — BOOLEAN

   — BREADTH

   — COMPLETION

   — CALL

   — CYCLE

   — DATA

   — DEPTH

— DEREF

— DICTIONARY

— EACH

ANSI Only–SQL3

— ELEMENT

— ELSEIF

— EQUALS

— GENERAL

— IF

— IGNORE

— INSTEAD

— LEAVE

— LESS

— LIMIT

ANSI Only–SQL3

— LIST

— LOOP

— MODIFY

— NEW

— NO

— NONE

— OFF

— OID

— OLD

— OPERATION

— OPERATORS
— OTHERS
— PARAMETERS
— PENDANT
— PREORDER
— PRIVATE
— PROTECTED
— RECURSIVE
— REF
— REFERENCING
— REPLACE
— RESIGNAL
— RETURN
— RETURNS
— ROLE
— ROUTINE
— ROW
— SAVEPOINT
— SEARCH
— SENSITIVE
— SEQUENCE
— SESSION
— SIGNAL
— SIMILAR
— SPACE
— SQLEXCEPTION
— SQLWARNING
— STRUCTURE
— TEST
— THERE

— TRIGGER

— TYPE

— UNDER

— VARIABLE

— VIRTUAL

— VISIBLE

— WAIT

— WHILE

— WITHOUT

Annex F
(Informative)

Maintenance and interpretation of SQL

ANSI Only–SQL3

ANSI Accredited Committee X3 provides formal procedures for revision, maintenance, and interpretation of ANSI Standards produced by X3. Section 5.2.3 of the Organization, Rules, and Procedures of X3 (X3/SD-2), ''Maintenance of American National Standards'', specifies procedures for defect management of ANSI X3 standards, including Errata, Amendments, and Interpretations. Errata and Interpretations are published by X3 in an SQL Information Bulletin. Amendments are processed under procedures that guarantee adequate public review before adoption.

Since publication of ANSI X3.135-1992, the following items have resulted in formal interpretations of Database Language SQL.

1) *to be provided*

The original questions and an explanation of the X3 interpretations, and correction of a number of SQL Errata, can be found in SQL Information Bulletin #*to be provided* (SQLIB-*to be provided*) published April 19, 1991, by Global Engineering Documents, Inc.

Since publication of ANSI X3.135-19892, several new defects have been discovered in the SQL language, leading to creation of the following defect reports against the equivalent International SQL standard, ISO/IEC 9075:1992. Numbers in parentheses refer to ISO/IEC JTC1/SC21/WG3 documents that identify the defects.

1) *to be provided*

The SQL language corrections proposed in each defect report were accepted by SC21/WG3 in *date to be provided* (see SC21 N*to be provided*, *city to be provided* WG3 Resolutions). These clarifications have all been endorsed by X3 technical committee X3H2. Formal processing within ANSI and further processing within ISO/IEC has been superseded by adoption of ISO/IEC 9075:199*x* as a replacement standard for ISO/IEC 9075:1992 and by adoption of ANSI X3.135-199*x* as a replacement standard for ANSI X3.135-1992.

All corrections to SQL proposed by these defect reports are included in this American National Standard.

X3H2-93-091 and YOK-003

Potential new questions or new defect reports addressing the specifications of this American National Standard should be communicated to:

X3 Secretariat, Computer and Business Equipment Manufacturers Association (CBEMA)
1250 Eye St., Suite 200
Washington, DC 20005.

ISO Only–SQL3

ISO/IEC JTC1 provides formal procedures for revision, maintenance, and interpretation of JTC1 Standards. Section 6.13 of the JTC1 Directives, ''Maintenance/correction of defects in JTC1 Standards'', specifies procedures for creating and processing ''defect reports''. Defect reports may result in technical corrigendum, amendments, interpretations, or other commentary on an existing International Standard. In addition, SC21, the JTC1 subcommittee that developed this International Standard, provides procedures for raising new ''questions'' about topics related to existing SC21 projects. Questions may result in interpretations, new project proposals, or possibly new defect reports.

Since publication of ISO/IEC 9075:1992, the following SC21 questions have resulted in formal interpretations of Database Language SQL. The first number in parentheses identifies the SC21 document in which the question was first raised, and the second number identifies the SC21 document in which the proposed interpretation was formally adopted.

1) *to be provided*

Since publication of ISO/IEC 9075:1992, several new defects have been discovered in the SQL language, leading to creation of the following defect reports.

1) *to be provided*

The SQL language corrections proposed in each defect report were accepted by SC21/WG3 in *date to be provided* (see SC21 N*to be provided*, *city to be provided* WG3 Resolutions). Further processing within SC21 was superseded by adoption of ISO/IEC 9075:199*x* as a replacement standard for ISO/IEC 9075:1992. All corrections to SQL proposed by these defect reports are included in this International Standard.

Potential new questions or new defect reports addressing the specifications of this International Standard should be communicated to:

Secretariat, ISO/IEC JTC1/SC21/WG3
Standards Council of Canada
Ottawa, Ontario
Canada.

Index

<1987> • ***18***
<1989> • ***18***
<1992> • ***18***

— A —

abandoned • 515, 516, 517, 518, 519, 520
ABSOLUTE • *100*, *617*, 618, 619
<abstract data type body> • 121, 135, 341, 399, *483*, **485**, 486, 487, 489, 492, *495*, 496, 541
<abstract data type definition> • 36, 70, 136, 141, 149, 170, 171, 273, 274, 341, 342, *355*, 356, 373, 376, 386, 481, **483**, 484, 486, 487, 488, 489, 490, 491, 492, 493, 497, 498, 521, 542, 543, 544, 550, *553*, 556, 582, 594, 595, 596, 612, 741, 835, 917, 919, 920
<abstract data type name> • 48, **117**, 124, *125*, *126*, 129, 147, 250, *333*, *343*, 344, 359, *362*, 364, 365, 366, 368, 369, 370, 376, *483*, *485*, 487, 488, 489, 490, 491, 492, *493*, *498*, 512, 520, 541, 582, 958
<abstract data type value expression> • ***184***, 185
ABSTRACT_DATA_TYPES • 801, 835, 861
ABSTRACT_DATA_TYPE_CATALOG • 794, 799, 801, 807, 808, 809, 835, 836, 837, 838, 839, 840, 861
ABSTRACT_DATA_TYPE_ENTRY • 801
ABSTRACT_DATA_TYPE_EQUALS • 801, 835
ABSTRACT_DATA_TYPE_ERROR • 801
ABSTRACT_DATA_TYPE_IMPACT_COLUMN • 808
ABSTRACT_DATA_TYPE_IMPACT_DOMAIN • 809
ABSTRACT_DATA_TYPE_LANGUAGE • 801, 835, 861
ABSTRACT_DATA_TYPE_LESS_THAN • 801, 835
ABSTRACT_DATA_TYPE_NAME • 794, 799, 801, 807, 808, 809, 835, 836, 837, 838, 839, 840, 861
ABSTRACT_DATA_TYPE_OPERATIONS • 801
ABSTRACT_DATA_TYPE_OPERATION_NAME • 801
ABSTRACT_DATA_TYPE_PARAMETERS • 801
ABSTRACT_DATA_TYPE_PRIVILEGES • 801, 807, 861
ABSTRACT_DATA_TYPE_RETURNS • 801
ABSTRACT_DATA_TYPE_SCHEMA • 794, 799, 801, 807, 808, 809, 835, 836, 837, 838, 839, 840, 861
access mode • 85, 91, 572, 626, 628, 633, 640, 644, 655, 656, 785, 786
ACTION • *100*, *389*, 390, 391, 396, *856*, 857, 918, 965
<action> • 82, ***343***, 347, 503, 504, 505, 512, 520, 917, 925
ACTOR • *100*, 376, *539*, 965
<actual identifier> • ***115***
Ada • 3, 4, 66, 79, 339, 536, 537, 545, 548, 557, 570, 747, 748, 749, 751, 753, 756, 760, 761, 762, 887, 911, 930, 940, 949, 958
ADA • *99*, *331*, 332, 339, 536, 544, 545, 550, 570, 572, 751, 835, *886*, 887, 958
<Ada assignment operator> • ***760***
<Ada host identifier> • *748*, ***760***, 761
<Ada initial value> • ***760***, 761
<Ada qualified type specification> • ***760***, 761, 762, 940
<Ada type specification> • ***760***, 762
<Ada unqualified type specification> • ***760***
<Ada variable definition> • *748*, **760**, 761, 762, 930
ADD • *100*, *403*, 409, *413*, *416*, *433*, 434, 437, *548*
<add column definition> • *401*, **403**, 404, 413, 936
<add domain constraint definition> • *430*, **433**, 926
<add supertable clause> • *401*, 402, **413**, 918
<add table constraint definition> • 387, *401*, 409, 414, **416**, 434, 437, 853, 936, 953
AFTER • 58, *100*, *459*, *464*, 467, 469, 474, 475, *631*, 634, 646, *881*, 882, 965
ALIAS • *100*, *631*, 633, 634, 636, 637, 921, 965
ALL • *100*, *149*, 151, 210, 227, *247*, 249, 252, 253, 254, 255, *265*, *298*, *307*, *343*, 344, 408, 420, 426, 436, 497, 503, 505, 520, *638*, 641, 643, 645, 646, *657*, *659*, 660, 664, *674*, *853*, *863*, 864, *891*, *892*, *893*, *894*, *898*, 932, 960
<all> • ***298***, 299
ALLOCATE • *100*, *692*, *722*
<allocate cursor statement> • 66, 70, 73, 75, 76, 77, 80, 123, *555*, 706, 708, **722**, 723, 726, 732, 733, 738, 927, 930, 952
<allocate descriptor statement> • 72, 78, 80, 123, *555*, **692**, 738
ALTER • *100*, *401*, *405*, 409, 411, 417, *430*, 434, 437, 519, 738
<alter column action> • ***405***
<alter column definition> • 379, *401*, **405**, 406, 407, 408, 936
<alter domain action> • ***430***

<alter domain statement> • 70, 398, **430**, 431, 432, 433, 434, 519, *553*, 738, 926
<alter table action> • ***401***, 402, 918
<alter table statement> • 69, 373, 375, 376, 381, 386, 387, 389, 398, **401**, 402, 404, 405, 410, 411, 413, 415, 416, 417, 519, *553*, 738, 935
ambiguous cursor name • 703, 719, 904
<ampersand> • *95*, **96**, *442*, *747*, 748
AND • 42, *100*, *205*, 206, *278*, 293, 294, 306, 386, *793*, *794*, *797*, *798*, *799*, *800*, *804*, *805*, *806*, *810*, *811*, *812*, *813*, *814*, *815*, *816*, *817*, *818*, *819*, *820*, *821*, *822*, *836*, *837*, *838*, *848*, *849*, *863*, *876*, *881*, *886*, *890*, *891*, *892*, *893*, *894*, *899*
ANY • *100*, 280, *298*, *659*, 660, *839*, *841*, *848*, *850*, *851*, *852*, *856*, *859*, *860*, *874*, *876*, *878*
applicable • 84, 129, 144, 147, 158, 170, 242, 253, 262, 263, 307, 325, 337, 342, 346, 348, 351, 376, 390, 391, 399, 409, 423, 424, 428, 430, 434, 437, 443, 449, 455, 460, 472, 492, 494, 496, 503, 508, 516, 518, 581, 612, 613, 626, 628, 632, 633, 640, 644, 653, 683, 711, 712, 713, 743, 750, 788, 839, 890, 891, 892, 893, 894, 896, 897
applicable privileges • 84, 129, 144, 147, 158, 170, 242, 253, 307, 337, 342, 346, 348, 351, 376, 390, 391, 399, 409, 423, 424, 428, 434, 437, 443, 449, 460, 472, 492, 496, 503, 518, 581, 626, 628, 632, 633, 640, 644, 683
application program • 2, 93
<approximate numeric literal> • ***107***, 110, 112, 176, 178
<approximate numeric type> • 29, ***126***, 128, 711, 713, 946, 947
<arc1> • ***18***
<arc2> • ***18***
<arc3> • ***18***
ARE • *100*, *535*, *748*
<argument> • 171, 325, ***333***, 334, 335, 336, 337, 338, 340, 396, 626, 629, 704, ***710***, 713, 714, 718
<argument list> • 325, ***333***, 334, 335, 336, 337
AS • *100*, *143*, 145, 146, *168*, 174, 175, 182, 200, 201, 222, 225, *237*, *238*, 256, 275, *333*, 336, 337, 340, *362*, *421*, 422, *427*, *439*, *442*, *464*, *465*, *485*, *486*, 490, *493*, *522*, *526*, *528*, 549, 573, 574, 575, 576, 582, 583, *652*, *669*, 705, 714, 715, *793*, *794*, *796*, *797*, *798*, *799*, *800*, *801*, *804*, *805*, *806*, *807*, *808*, *809*, *810*, *811*, *812*, *813*, *814*, *815*, *816*, *817*, *818*, *819*, *820*, *821*, *822*, *823*, *824*, *825*, *826*, *827*, *828*, *829*, *853*, *890*, *891*, *892*, *893*, *894*, *895*, *896*, *897*, *898*, *899*, 932
ASC • *100*, *609*, 788
<as clause> • ***237***, 239, 957
ASSERTION • *100*, 358, *459*, *463*, 519, *792*, *899*, *900*
<assertion column list> • ***459***, 460, 461
<assertion definition> • 69, *355*, 357, **459**, 460, 461, *553*, 615, 622, 627, 630, 636, 642, 647, 738, 858, 919, 925, 926, 927, 929, 931
ASSERTIONS • 814, 858, 862, 898, 931
<assertion trigger> • ***459***, 461, 919
<assertion trigger statement> • ***459***
assignable • 8, 38, 39, 40, 41, 170, 317, 592
<assignment operator> • *98*, **99**, *379*
<assignment statement> • 72, 77, 141, 142, *554*, **592**, 593, 738, 921
<assignment target> • 141, 142, ***592***, 593
<asterisk> • *95*, **96**, *99*, 103, 104, *149*, *187*, 188, *199*, *237*, *286*, 288, 289, 290, 292, *296*, *455*, *771*
ASYNC • *100*, *553*, 556, 571, 577, 920, 965
asynchronous • 79, 571, 659, 660, 666, 904, 906, 907, 950
asynchronous SQL statement not accepted • 79, 571, 904
asynchronous SQL statement returned an SQLSTATE value • 660, 904
<async statement identifier> • ***553***, 571, *659*, 660
<async statement identifier list> • ***659***, 660
AT • *100*, *196*, 197
ATOMIC • *99*, *584*, 585, 586
<attribute reference> • *133*, **141**, 142, *592*, 915
AUTHORIZATION • *100*, *355*, 357, *535*, 538, *547*, *681*, 741, *790*, *831*, *865*, *890*, *893*, *894*, *895*, *896*, 935, 949
<authorization identifier> • 58, 59, 60, 82, 83, 84, 90, **115**, 121, 122, 137, *344*, 346, *355*, 356, 358, 363, 368, 370, 402, 408, 409, 411, 419, 420, 422, 426, 428, 430, 434, 436, 437, 438, 439, 441, 443, 444, 445, 449, 450, 452, 455, 456, 457, 460, 463, 472, 478, 481, 483, 493, 494, 496, 497, 498, 500, 502, 503, 504, 505, 507, 508, 510, 511, 512, 516, 518, *535*, 537, 551, 570, 650, 651, 653, 669, 670, 681, 683, 743, 744, 784, 832, 861, 863, 864, 865, 868, 870, 873, 889, 945, 946, 949, 951, 956
AVG • *100*, *149*, 150, 151, 947

— B —

<1989 base> • ***18***
based on • 3, 17, 25, 43, 51, 93, 198, 274, 366, 405, 411, 434, 488, 493, 495, 597, 599, 702, 722
BEFORE • 58, *100*, *459*, *464*, 467, 469, 474, 476, *631*, 634, 646, *881*, 882, 965
BEGIN • *100*, *338*, *490*, *491*, *584*, *604*, 739, *748*
<beginning label> • ***584***, 585, *602*, *604*, 649, 650
BETWEEN • *100*, *278*, 294
<between predicate> • *271*, **278**, 703, 705
BIT • 21, 40, *100*, *125*, 127, 573, 574, 575, 576, 687, 690, 700, 704, *760*, 761, *763*, 764, 765, *767*, 768, *771*, 772, 773, *776*, 777, 778, *779*, 780, 781, *836*, 927, 930, 931
<bit> • *106*, **107**, 109
<bit concatenation> • 28, ***190***, 191, 192, 193
<bit factor> • ***190***, 191, 192
<bit length expression> • ***152***, 154, 923
<bit primary> • ***190***, 191, 192

bit string data type descriptor • 27
<bit string literal> • *98*, 103, ***106***, 109, 110, 112, 113, 380, 381, 922, 923
<bit string type> • ***125***, 127, 132, 711, 713, 923
<bit substring function> • 28, ***155***, 157, 160, 161
<bit value expression> • *155*, 157, 160, ***190***, 191, 192, 193, 924
<bit value function> • ***155***, 161, 923
BIT_LENGTH • *100*, *152*, 154, 176, 179, 180, 704, 765
BOOLEAN • 21, 30, *100*, *126*, *489*, *522*, *523*, *526*, *545*, 557, 559, 561, 564, 566, 567, 569, 573, 574, 575, 576, 577, 690, *760*, 762, 765, 768, 772, *776*, 777, 780, *838*, 965
<boolean factor> • 43, ***205***
<boolean literal> • *106*, **108**, 113, 380, 915
<boolean predicate> • *271*, **313**, 314
<boolean primary> • ***205***, 315, 917
<boolean term> • ***205***
<boolean type> • *125*, **126**, 131, 712, 713
<boolean value expression> • *184*, 185, 186, ***205***, 206, *313*, *315*, 492, 915, 916, 917
BOTH • *100*, *155*, 157, 160, 172, 180, 181, 182, 669, 670, 677, 678, 680, 681, 692, 706, 722
<bracketed comment> • ***99***, 104
<bracketed comment contents> • ***99***, 104
<bracketed comment introducer> • ***99***, 103, 104
<bracketed comment terminator> • ***99***, 103
BREADTH • *100*, *260*, 262, 263, 264, 965
BY • *100*, *233*, *248*, *260*, *609*, 610, 613, 721, 787, *801*, *898*, *899*, 927, 930

— C —

C • 3, 4, 66, 79, *99*, *331*, 332, 339, 544, 550, 559, 573, 711, 712, *747*, *748*, 749, 751, 753, 756, *763*, 764, 765, 766, 835, *886*, 887, 905, 911, 930, 940, 949, 950
CALL • *100*, *547*, *581*, 739, 965
<caller language clause> • 62, *539*, **540**, 541, 545, 549, 550, 552, 570, 572, 573, 574, 575, 576, 577
<call statement> • 62, 71, 77, *554*, **581**, *702*, 718, 739, 920
candidate key • 48, 51, 52, 53, 57, 240
cardinality • 8, 44, 150, 151, 173, 209, 219, 238, 243, 263, 266, 268, 300, 623, 634, 741, 792, 855, 904, 959
cardinality violation • 173, 263, 266, 268, 623, 634, 904
CARDINAL_NUMBER • 829, 836, 848, 854, 867, 874, 883
<C array specification> • ***763***, 764, 765
CASCADE • 86, *100*, *358*, 359, *389*, 391, 393, 394, 410, 411, 417, 419, 420, 426, 436, 437, 438, 445, 452, 453, 457, 469, 481, 497, 498, 499, 518, 519, 520, *614*, 615, 633, 640, 644, *856*, 857, 919
CASCADED • 48, *100*, 245, *421*, 422, 424, 425, *846*, 847, 926
CASE • 92, *100*, *165*, 166, *597*, *798*, *799*
<case abbreviation> • ***165***
<case expression> • 27, **165**, 166, 167, *184*, 185, 186, 323, 932, 933
case not found for case statement • 598, 904
<case operand> • ***165***, 166
<case specification> • ***165***, 166, 703, 705
<case statement> • 72, *554*, **597**, 598, 920
<case statement else clause> • ***597***, 598
<case statement when clause> • ***597***
CAST • 41, *100*, *168*, 170, 174, 175, 182, 200, 201, 239, 256, 275, 325, 336, 337, 340, *486*, 490, *522*, *526*, *528*, *540*, *546*, 549, 573, 574, 575, 576, 582, 583, 705, 714, 715, 907
<cast clause> • 170, 171, *363*, *485*, **486**, 489, 490
<cast function> • ***486***
<cast operand> • ***168***, 171
<cast specification> • 51, 66, **168**, 169, 171, 174, 175, 183, *184*, 185, 186, 714, 715, 932, 933
<cast target> • ***168***
CATALOG • *100*, *677*, 678, 740
<catalog name> • 17, 59, 81, ***115***, 120, 121, 123, 124, 345, 356, 358, 480, 481, 536, 677, 678, 742, 743, 744, 923, 944, 945, 946, 949
CATALOG_NAME • *99*, *547*, *735*, 737, 742, 743, *791*, *792*, *793*, *794*, *796*, *797*, *798*, *799*, *800*, *804*, *805*, *806*, *810*, *811*, *812*, *813*, *814*, *815*, *816*, *817*, *818*, *819*, *820*, *821*, *822*, *823*, *824*, *825*, *826*, *834*, *839*, *841*, *848*, *850*, *851*, *852*, *856*, *859*, *860*, *874*, *876*, *878*
<C bit variable> • ***763***, 764, 765, 766, 930
<C character variable> • ***763***, 764, 765
<C class modifier> • ***763***
<C derived variable> • ***763***, 766, 930, 940
CHAR • *100*, *125*, 127, 548, *760*, 761, 762, *776*, 777, *779*, 781

character • 3, 4, 7, 8, 9, 10, 11, 12, 18, 21, 22, 23, 24, 25, 28, 29, 31, 35, 40, 42, 43, 58, 59, 61, 64, 65, 66, 68, 70, 81, 82, 83, 90, 91, 92, 93, 95, 96, 97, 98, 99, 103, 104, 105, 106, 109, 110, 112, 114, 115, 117, 118, 121, 122, 123, 124, 125, 127, 130, 132, 135, 136, 137, 147, 150, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 169, 170, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 185, 190, 191, 192, 193, 233, 235, 241, 252, 262, 273, 274, 275, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 317, 318, 320, 321, 323, 339, 343, 344, 348, 349, 350, 355, 356, 357, 358, 359, 375, 377, 379, 380, 382, 383, 399, 425, 427, 428, 442, 443, 444, 445, 446, 447, 448, 449, 450, 452, 454, 455, 456, 457, 461, 467, 504, 505, 512, 513, 514, 516, 517, 518, 519, 520, 529, 535, 536, 538, 548, 550, 553, 558, 559, 560, 561, 563, 564, 565, 566, 567, 568, 569, 570, 573, 574, 575, 576, 577, 589, 611, 637, 642, 648, 669, 677, 678, 680, 681, 688, 689, 692, 697, 700, 702, 704, 706, 711, 712, 716, 719, 722, 737, 738, 739, 742, 744, 747, 748, 749, 751, 753, 754, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 815, 816, 817, 828, 834, 839, 841, 846, 849, 858, 873, 874, 875, 876, 877, 878, 879, 884, 887, 903, 904, 905, 907, 918, 922, 923, 931, 932, 933, 934, 935, 936, 937, 938, 940, 943, 945, 946, 947, 948, 949, 950, 952, 953, 955, 957, 958, 960, 961
CHARACTER • 21, 22, 23, 40, 66, *100*, *125*, 127, 132, 135, *343*, 348, *355*, 359, 375, 380, 427, *442*, 443, *445*, *489*, 506, *522*, *523*, *528*, 573, 574, 575, 576, 577, *604*, 688, 690, 700, 704, 706, 739, *760*, 761, *763*, 764, 765, *767*, 768, *771*, 772, 774, *776*, 777, *779*, 780, 781, *828*, *836*, *872*, 873, 932, 936, 937, 940, 943, 946
<character enumeration> • ***287***, 289, 292, 294
<character factor> • ***190***, 191, 192
<character list> • ***442***, 444, 918
character not in repertoire • 130, 904
<character primary> • ***190***, 192
<character range> • ***447***, 448
character repertoire • 8, 10, 22, 23, 25, 40, 61, 91, 103, 109, 110, 118, 127, 130, 137, 152, 156, 157, 159, 169, 190, 275, 323, 348, 349, 379, 443, 448, 449, 452, 520, 765, 874, 949, 957
<character representation> • 105, ***106***, *109*, 110, 112, 114, 284, *286*, *287*, 291, 448, *760*, 761, *764*, *767*, 768, *779*, 780, 922
<character set> • 81, *286*, **287**
<character set definition> • 70, *355*, 357, **442**, 443, 444, 457, *553*, 739, 935, 936
character set descriptor • 23, 59, 349, 443, 452, 518, 520, 874
<character set name> • 58, 110, **117**, 122, 123, 124, 127, *343*, 344, *348*, 359, 428, *442*, 443, *445*, 450, 456, 504, 512, 514, 680, 711, 712, 945, 946
<character set source> • ***442***, 444, 918
<character set specification> • 103, 104, *106*, 109, 110, 114, *115*, 118, 124, *125*, 127, **348**, 349, 350, *355*, 356, 375, *446*, 448, *454*, 505, 519, *535*, 688, *748*, *760*, 761, 762, *763*, 765, 766, *767*, 768, 770, *771*, 772, 773, *776*, 777, 778, *779*, 780, 781, 923, 934, 940, 949, 952
<character set specifier> • ***287***
<character specification> • ***442***, 443, *447*
<character specifier> • *286*, ***287***, 288, 289, 292
character string data type descriptor • 22
<character string literal> • 90, *98*, 103, 104, ***106***, 109, 110, 112, 114, 118, 379, *442*, 443, 467, *589*, 680, 923
<character string type> • ***125***, 127, 132, 375, 427, 711, 712, 931, 946
<character substring function> • 24, 28, ***155***, 156, 157, 158, 161, 704, 932
<character translation> • 24, ***155***, 156, 159, 161, 704, 923, 948
<character value expression> • 25, *152*, 153, *155*, 156, 157, 158, 159, 160, ***190***, 191, 192, 193, *282*, *286*, 287, 288, 289, 291, 292, 704, 933
<character value function> • ***155***, 157, 161, 923, 932
CHARACTER_DATA • 828, 835, 836, 841, 843, 845, 846, 848, 852, 856, 858, 861, 862, 863, 865, 868, 870, 872, 876, 881, 883, 886
CHARACTER_LENGTH • *100*, *152*, 179, 704
CHARACTER_MAXIMUM_LENGTH • 794, 799, 836, 837, 838, 839
CHARACTER_OCTET_LENGTH • 794, 799, 836, 837, 838, 839
CHARACTER_SETS • 815, 872, 874, 876, 878, 931
CHARACTER_SET_CATALOG • *99*, 688, 689, *695*, 697, 700, 701, 711, 712, 714, 715, *793*, *794*, *799*, *815*, *816*, *817*, *834*, *872*, *874*, 875, *876*, 877, *878*, 879, 952
CHARACTER_SET_NAME • *99*, *547*, 688, 689, *695*, 697, 700, 701, 711, 712, 714, 715, *793*, *794*, *799*, *815*, *816*, *817*, *834*, *872*, *874*, 875, *876*, 877, *878*, 879, 952
CHARACTER_SET_SCHEMA • *99*, 688, 689, *695*, 697, 700, 701, 711, 712, 714, 715, *793*, *794*, *799*, *815*, *816*, *817*, *834*, *872*, *874*, 875, *876*, 877, *878*, 879, 952
<char length expression> • ***152***, 154
CHAR_LENGTH • *100*, *152*, 704
CHECK • 48, 92, *100*, 245, 375, 376, 386, *398*, *421*, 422, 424, 425, *459*, *526*, *548*, 636, 641, 646, 743, *791*, *792*, *798*, *812*, *820*, *821*, *829*, *832*, *833*, *836*, *839*, *841*, *843*, *845*, *846*, 847, *848*, *850*, *851*, *852*, *853*, *854*, *856*, *858*, *859*, *860*, *861*, *862*, *863*, *865*, *868*, *870*, *872*, *874*, *876*, *878*, *881*, *883*, *886*, 887, *898*, *899*, *900*, 931

<check constraint definition> • *372*, *373*, 376, *384*, 385, 386, **398**, 399, 409, *427*, 434, 437, 486, 858, 926
CHECK_CLAUSE • 812, 858
CHECK_COLUMN_USAGE • 820, 821, 860
CHECK_CONSTRAINTS • 812, 843, 853, 858, 859, 860, 862, 931
CHECK_OPTION • 548, 798, 846, 847
CHECK_TABLE_USAGE • 859
CHECK_TIME • 862
<C host identifier> • *748*, ***763***, 764, 765
<C initial value> • ***763***, 764
<circumflex> • ***96***, *287*, 288, 289, 290, 292
CLASS • *99*, *100*, *343*, 359, *439*, *441*, *545*, *546*, *547*, *548*, 689, 701, 711, 712, *735*, 737, 742, *794*, *799*, *802*, *803*, *841*, 842, *848*, *849*, *866*, *867*, *872*, 873, 952
CLASS_ORIGIN • *99*, *100*, *735*, 737, 742, 952
CLOSE • *100*, *621*, 664, 667, *728*, 739
<close statement> • 58, 66, 71, 73, 75, 76, 77, 396, 425, 477, *554*, 578, **621**, 627, 739, 754
cluster • 60, 830
<C numeric variable> • ***763***, 765
COALESCE • 9, *100*, *165*, 166, 222, 225, 703, 705, *799*
COBOL • 3, 4, 66, 79, *99*, *331*, 332, 339, 544, 550, 561, 574, *747*, *748*, 749, 751, 753, 756, *767*, 768, 769, 770, 835, *886*, 887, 888, 912, 930, 940, 949, 950, 952, 963
<COBOL binary integer> • ***767***, 769, 940
<COBOL bit type> • ***767***, 768, 769, 930
<COBOL character type> • ***767***, 768, 769, 770, 940
<COBOL computational integer> • ***767***, 769, 963
<COBOL host identifier> • *748*, ***767***, 768
<COBOL integer type> • ***767***, 940
<COBOL nines> • ***767***
<COBOL nines specification> • ***767***
<COBOL numeric type> • ***767***, 769
<COBOL type specification> • ***767***, 769, 930, 952
<COBOL variable definition> • *748*, **767**, 768, 769, 952
coercibility • 8, 25, 26, 40, 136, 147, 150, 156, 157, 169, 185, 191, 233, 275, 282, 288, 323, 376, 421, 422, 611
Coercible • 25, 26, 27, 136, 169, 275, 376
COLLATE • *100*, *351*, 703, *815*, *874*, 875, 961
<collate clause> • 23, 25, 41, 43, *190*, 191, 193, *233*, 234, **351**, *372*, 375, 376, 377, *427*, 428, *442*, 443, 518, *609*, 611, 924, 925, 926
<collating basis> • ***447***, 449, 450
<collating chars> • ***447***, 448, 449, *455*, 456
<collating foundation> • ***447***, 450
<collating modifier> • ***447***, 449
<collating modifiers> • ***447***, 449
<collating sequence definition> • ***446***, *447*, 448
COLLATION • *100*, *343*, 359, *442*, 443, *446*, 448, 449, 450, *452*, 505
<collation definition> • 70, *355*, 357, 443, **446**, 447, 448, 449, 450, 513, *553*, 739, 925, 926, 927, 929, 931, 949
collation descriptor • 23, 59, 445, 447, 450, 452, 457, 505, 513, 518, 520, 876
<collation dictionary specification> • ***446***, 449, 450, 918
<collation name> • 23, 41, 43, **117**, 122, 123, 124, *343*, 344, *351*, 359, 377, 409, 428, 437, *446*, *447*, 448, 449, 450, *452*, 504, 512, 513, 514, 703, 711, 712, 923
<collation option> • ***447***
<collation options> • ***447***, 448
<collation routine specification> • ***446***, 450, 918
COLLATIONS • 794, 800, 816, 839, 872, 874, 876, 931
<collation source> • 23, *442*, 443, 444, ***446***, 448, 449, 450, 518, 918
COLLATION_CATALOG • *99*, 689, *695*, 697, 699, 711, 712, *794*, *799*, *800*, *816*, *836*, *837*, *838*, *839*, *872*, *874*, *876*, 877, 952
COLLATION_DEFINITION • 816, 876, 877, 953
COLLATION_DICTIONARY • 816, 876, 877
COLLATION_NAME • *99*, 689, *695*, 697, 699, 711, 712, *794*, *799*, *800*, *816*, *836*, *837*, *838*, *839*, *872*, *874*, *876*, 877, 952
COLLATION_SCHEMA • *99*, 689, *695*, 697, 699, 711, 712, *794*, *799*, *800*, *816*, *836*, *837*, *838*, *839*, *872*, *874*, *876*, 877, 952
COLLATION_TYPE • 816, 876, 877
<collection element> • ***211***, 212, 213, 214, 215
<collection expression> • ***248***, 249, 250, 254
<collection list> • ***211***, 212, *213*, 214, *215*
<collection literal> • *106*, **109**, 112, 113
<collection type> • 38, *125*, **126**, 131, 150, 185, 249, 250, 612
<collection value expression> • ***184***, 185, 186
<colon> • *95*, **96**, *107*, *108*, *117*, *287*, 289, 292, 293, 329, *584*, *602*, *604*, *748*, *760*, *776*

column • 17, 23, 25, 36, 38, 41, 42, 43, 44, 45, 47, 48, 50, 51, 52, 53, 54, 55, 56, 57, 58, 67, 80, 81, 82, 83, 92, 116, 122, 127, 141, 143, 144, 145, 146, 147, 148, 149, 150, 151, 164, 184, 185, 208, 209, 210, 217, 219, 221, 222, 223, 224, 225, 226, 227, 228, 229, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 248, 249, 250, 251, 252, 253, 256, 259, 260, 261, 262, 263, 264, 265, 266, 268, 275, 284, 297, 298, 301, 302, 305, 306, 307, 315, 320, 322, 336, 343, 344, 346, 347, 356, 363, 364, 365, 366, 367, 368, 369, 370, 372, 373, 374, 375, 376, 377, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 399, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 415, 417, 421, 422, 423, 424, 427, 428, 430, 432, 434, 436, 437, 441, 445, 452, 459, 460, 461, 465, 466, 467, 469, 470, 471, 472, 473, 475, 476, 477, 478, 483, 485, 486, 487, 488, 491, 492, 498, 503, 504, 505, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 544, 545, 551, 582, 592, 595, 597, 604, 609, 610, 611, 612, 613, 617, 618, 619, 620, 624, 626, 628, 629, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 644, 645, 646, 647, 648, 650, 651, 652, 653, 681, 696, 697, 705, 706, 709, 710, 711, 715, 721, 726, 730, 743, 787, 789, 799, 800, 805, 808, 813, 819, 821, 822, 826, 834, 836, 839, 840, 846, 848, 849, 851, 854, 855, 857, 858, 860, 861, 863, 864, 865, 869, 870, 871, 882, 885, 887, 892, 894, 900, 903, 915, 916, 917, 918, 921, 922, 923, 925, 927, 930, 932, 933, 934, 935, 936, 937, 938, 953, 956, 957, 958, 959
COLUMN • *100*, *403*, *405*, 408, *410*, 434, *436*, 437, 519
<column constraint> • 364, *372*, **373**, 375, 377, 650, 935
<column constraint definition> • 366, 367, ***372***, 374, 375, 376
<column definition> • 127, 356, *363*, 364, 365, 366, 367, 369, 370, **372**, 373, 375, 376, 377, 379, 387, 388, *403*, 404, 460, *485*, 486, 487, 488, 491, 498, 521, 633, 650, 651, 917, 935
column descriptor • 36, 43, 44, 47, 48, 208, 233, 239, 249, 250, 251, 252, 366, 367, 377, 381, 382, 405, 406, 407, 408, 413, 422, 428, 432, 434, 436, 437, 452, 483, 491, 492, 517, 518, 519, 520, 611, 624, 629, 632, 639, 644, 650, 653, 710, 848, 849
<column name> • 42, **116**, 122, *141*, *143*, 144, *146*, 209, 222, 223, 225, 233, *237*, 238, 239, 242, 250, 251, 252, 253, *260*, 261, 264, 344, 346, *363*, 364, 365, 366, 367, 368, *372*, 373, 376, 377, 386, 387, 388, 389, 390, 399, 404, *405*, 406, 407, 408, 409, *410*, 413, 417, 421, 422, 423, 424, 434, 437, 460, 466, 467, 469, 470, 471, 472, 473, 475, 478, 487, 488, 491, 492, 505, 513, 514, 515, 516, 519, 592, 604, 611, 624, 629, 632, 633, 635, 636, 637, *638*, 639, 640, 642, 644, 647, 651, 652, 653, 730, 743, 787, 915, 935, 938, 957, 958
<column name list> • 82, 83, ***143***, *221*, *248*, *260*, *344*, 366, *386*, *389*, *421*, *459*, *465*, *609*, 610, 611, 613, *631*, 632, 639, 721, 730, 927, 930
<column reference> • 23, 53, 54, 56, **146**, 147, 148, 149, 151, *164*, *184*, 185, 217, 222, 231, *233*, 235, 236, 238, 239, 241, 246, 261, 284, 297, 315, 376, 403, 428, 611, 612, 637, 846, 858, 916, 923, 932, 934, 957
COLUMNS • 794, 799, 808, 822, 826, 839, 845, 848, 851, 854, 860, 863, 870, 885, 931
COLUMN_DEFAULT • 799, 848, 849
COLUMN_DOMAIN_USAGE • 822, 931
COLUMN_NAME • *99*, *736*, 737, 743, *794*, *799*, *800*, *805*, *808*, *813*, *819*, *821*, *822*, *836*, *838*, *839*, *841*, *848*, *849*, *851*, *854*, *860*, *863*, 864, *870*, 871, *891*, *892*, *893*, *894*, *899*, *900*
COLUMN_PRIVILEGES • 797, 800, 805, 870, 892, 931
combination row • 262, 264, 265
<comma> • *95*, **96**, *126*, *143*, *165*, *203*, *207*, *210*, *211*, *219*, *233*, *237*, *260*, *280*, *307*, *327*, *333*, *341*, *343*, *363*, *439*, *442*, *447*, *454*, *455*, *459*, *465*, *480*, *485*, *495*, *503*, *508*, *510*, *512*, *539*, *582*, *587*, *590*, *609*, *617*, *623*, *638*, *655*, *657*, *659*, *695*, *699*, *710*, *735*, *760*, *763*, *771*, *774*, *776*, *779*
COMMAND_FUNCTION • *99*, 606, *735*, 737, 738, 744
<comment> • ***99***, 104
<comment character> • ***99***
COMMIT • 86, 87, 88, *99*, *100*, *361*, 366, 390, 398, *459*, *649*, 650, *655*, *664*, 739
<commit statement> • 49, 58, 66, 69, 71, 85, 87, 89, 353, 396, 425, 461, 477, 537, *554*, 577, 584, 627, **664**, 674, 739, 784, 958
COMMITTED • 86, 87, *99*, *100*, *655*
common column name • 146, 222, 223
comparable • 8, 23, 28, 30, 32, 33, 34, 35, 38, 39, 40, 157, 166, 200, 222, 225, 261, 273, 275, 278, 282, 287, 298, 302, 305, 310, 317, 320, 323, 390, 597
<comparison predicate> • 186, 269, *271*, **273**, 276, 298, 299, 613, 703, 705, 788, 933, 934
compilation unit • 1, 2, 61, 62, 79, 80, 85, 748, 761, 764, 768, 771, 774, 776, 779, 944
COMPLETION • *100*, *545*, *547*, *659*, 664, 741, 965
<component name> • **117**, 124, *485*, 487, 488

<component renaming clause> • ***485***, 487
<component renaming element> • ***485***
<comp op> • ***273***, 274, 275, 276, 277, *298*, 299, 612, 613, 788
<compound statement> • 46, 61, 62, 64, 66, 72, 77, 119, 139, 162, *237*, 244, 245, 257, 334, 338, *554*, 555, **584**, 585, 586, 587, 588, 590, 609, 649, 650, 739, 920
<concatenation> • ***190***, 191, 192, 193, 933
<concatenation operator> • 24, *98*, **99**, *190*, 704
<condition> • ***755***, 756, 963
<condition action> • ***755***, 757
<condition information> • ***735***, 741
<condition information item> • ***735***, 736, 741
<condition information item name> • ***735***, 736, 737
<condition number> • *735*, **736**, 741, 960
CONDITION_NUMBER • *99*, *547*, *735*, 737, 741
conformance • 12, 15, 18, 92, 93, 827, 886, 887, 911, 912, 913
Conformance • 911
CONNECT • *100*, 571, *669*, 785
CONNECTION • *99*, *100*, *545*, *547*, 571, *672*, *735*, 737, 740, 744, 785
connection exception • 89, 571, 670, 672, 674, 785, 904
connection failure • 89, 672, 904
<connection name> • 88, **117**, 122, 123, 124, *669*, 670, *672*, 674, 744
connection name in use • 670, 904
<connection object> • ***672***, *674*
<connection target> • ***669***
CONNECTION_NAME • *99*, *545*, *547*, *735*, 737, 744
<connect statement> • 72, 83, 88, 89, 90, 91, *554*, 571, **669**, 670, 671, 739, 744, 785, 928, 944, 945, 950, 951, 956, 960
consistent • 472, 717, 961
<constant or updatable> • ***362***, 370, *372*, 377, *485*, *590*
CONSTRAINT • *100*, *221*, 224, 225, 226, 228, *352*, 408, *417*, *434*, *436*, 437, 519, 626, 628, 633, 639, 644, 740, *755*, 756, 757, 759, *791*, *829*, *832*, *833*, *834*, *835*, *836*, *838*, *839*, *841*, *843*, *845*, *846*, *848*, *850*, *851*, *852*, *853*, *854*, *856*, *858*, *859*, *860*, *862*, *868*, *870*, *872*, *874*, *876*, *878*, *886*, 922
<constraint attributes> • **352**, 353, *372*, 375, *384*, 409, *427*, 434, 437, *459*, 925
<constraint check time> • ***352***
<constraint disposition> • *408*, *434*, 435, ***436***, 438
<constraint join> • ***221***, 230, 916
constraint mode • 49, 85, 91, 352, 353, 384, 460, 461, 572, 627, 630, 636, 642, 647, 657, 785
<constraint name> • **117**, 123, 124, *221*, 224, *352*, 358, 384, 401, 408, 409, *417*, 427, *434*, 436, 437, *459*, 460, *463*, 519, 615, 622, *657*, *755*, 757, 853, 898, 931, 958
<constraint name definition> • **352**, 353, *372*, 375, 378, *384*, 385, 409, *427*, 434, 437, 935, 953, 958
<constraint name list> • *408*, *436*, 437, ***657***
constraint-referential-action graph • 615, 622, 633, 644, 645
CONSTRAINTS • *100*, 615, 622, *657*, 664, *796*, *810*, *811*, *812*, *820*, *821*, *843*, *852*, *853*, *854*, *856*, *858*, *859*, *860*, *862*, *898*, *899*, *900*, 931
CONSTRAINT_CATALOG • *99*, *735*, 737, 742, 743, 744, *796*, *810*, *811*, *812*, *813*, *814*, *820*, *821*, *843*, *852*, *853*, *854*, *856*, 857, *858*, *859*, *860*, *862*, *898*, *899*, *900*
CONSTRAINT_COLUMN_USAGE • 821, 931
CONSTRAINT_NAME • *99*, *735*, 737, 742, 743, 744, *796*, *810*, *811*, *812*, *813*, *814*, *820*, *821*, 841, *843*, *852*, *853*, *854*, *856*, 857, *858*, *859*, *860*, *862*, *898*, *899*, *900*, 953
CONSTRAINT_SCHEMA • *99*, *735*, 737, 742, 743, 744, *796*, *810*, *811*, *812*, *813*, *814*, *820*, *821*, *843*, *852*, *853*, *854*, *856*, 857, *858*, *859*, *860*, *862*, *898*, *899*, *900*
CONSTRAINT_TABLE_USAGE • 820, 931
CONSTRAINT_TYPE • 810, 852, 853, 854, 856, 900
CONSTRUCTOR • *100*, *490*, *531*, *532*, *533*, *539*, 541, 594
CONTINUE • *100*, *755*
CONVERT • *100*, *155*
Coordinated Universal Time • 7
<correlation name> • **116**, 121, 122, *143*, 145, *146*, 147, 240, 245, *260*, 261, 262, 263, 264, 266, 307, *465*, *466*
<correlation name list> • ***260***, 261, 262
CORRESPONDING • *100*, *248*, 250, 251, 259, 410, 934
<corresponding column list> • ***248***, 251
corresponding join columns • 222, 226
<corresponding spec> • *247*, **248**
COUNT • 80, *100*, *149*, 150, 151, 239, 386, 391, 523, *546*, *695*, 696, *699*, 700, 710, 712, 713, 715, *735*, 737, 741, *791*, *792*, *853*, *898*, *899*, 932, 947, 960
counterpart • 53, 54, 55, 56, 240, 245, 246, 423
CREATE • *100*, *355*, *361*, *421*, *427*, *439*, *442*, *446*, *454*, *459*, *464*, 466, *483*, *493*, *495*, *500*, *507*, *522*, *526*, *528*, *531*, *532*, *533*, *539*, 540, *652*, 738, 739, 740, 741, *790*, *791*, *792*, *793*, *794*, *796*, *797*, *798*, *799*, *801*, *802*, *803*, *804*, *805*, *806*, *807*, *808*, *809*, *810*, *811*, *812*, *813*, *814*, *815*, *816*, *817*, *818*, *819*, *820*, *821*, *822*, *823*, *824*, *825*, *826*, *827*, *828*, *829*, *831*, *832*, *833*, *834*, *835*, *836*, *841*, *843*, *845*, *846*, *848*, *850*, *851*, *852*, *854*, *856*, *858*, *859*, *860*, *861*, *862*, *863*, *865*, *866*, *867*, *868*, *870*, *872*, *874*, *876*, *878*, *880*, *881*, *883*, *885*, *886*, *889*, *890*, *891*, *892*, *893*, *894*, *895*, *896*, *897*, *898*, *899*, *900*, *901*
created local temporary table • 45, 46, 47, 90, 358, 361, 369, 390, 398, 945, 956
CROSS • *100*, *221*, 223
<cross join> • 54, ***221***, 226, 227, 230, 924
<C storage class> • ***763***
CURRENT • *100*, *625*, *638*, *674*, *729*, *730*, *732*, *733*
<current date value function> • ***162***

current SQL-connection • 88, 89, 670, 672, 673, 674, 675
current SQL-session • 59, 84, 89, 90, 110, 118, 120, 123, 135, 363, 436, 438, 472, 478, 481, 496, 497, 498, 500, 502, 507, 510, 511, 551, 652, 653, 657, 670, 672, 677, 678, 680, 681, 683, 684, 744, 784, 865, 889, 890, 891, 892, 944, 945
<current timestamp value function> • ***162***
<current time value function> • ***162***
CURRENT_DATE • 87, *100*, *162*, 181, 935, 957
CURRENT_TIME • *100*, *162*, 935, 957
CURRENT_TIMESTAMP • *100*, *162*, 935, 957
CURRENT_USER • 87, 93, *100*, *133*, 135, 137, 138, *379*, 380, 381, 382, 383, 398, 459, 615, 787, *793*, *794*, *796*, *797*, *798*, *800*, *804*, *805*, *806*, *810*, *811*, *812*, *813*, *814*, *815*, *816*, *817*, *818*, *819*, *820*, *821*, *822*, 932, 935, 947
CURSOR • *100*, 604, *609*, *721*, *722*
cursor mode • 58, 86, 392, 395, 396, 425, 477, 578, 614, 615, 618, 621, 622, 626, 627, 640, 657, 664, 754, 907
<cursor name> • 80, ***116***, *117*, 119, 120, 122, 139, 536, 585, 604, *609*, 610, *614*, *617*, 618, *621*, *625*, *631*, 632, *638*, 639, 703, 719, *721*, 724, 726, 728, 729, 730, 731, *732*, *733*, 749, 750, 928, 959
cursor operation conflict • 396, 626, 629, 641, 645, 742, 908
cursor sensitivity exception • 616, 626, 629, 633, 634, 640, 645, 904
<cursor specification> • 80, 144, 147, ***609***, 610, 611, 612, 613, 614, 617, 622, 625, 626, 632, 639, *702*, 707, 708, 717, 721, 722, 724, 726, 729, 730, 732, 733, 928, 930, 937
cursor specification cannot be executed • 717, 906
CURSOR_NAME • *99*, *545*, *547*, *736*, 737, 742, 743
<C VARCHAR variable> • ***763***, 764, 765, 766, 940
<C variable definition> • *748*, **763**, 764, 765, 766
<C variable specification> • ***763***
CYCLE • *100*, *260*, 965
<cycle clause> • ***260***, 261, 263, 265
<cycle column> • ***260***, 263, 265
<cycle column list> • ***260***, 261
<cycle mark column> • ***260***, 262, 263, 265
<cycle mark value> • ***260***, 262, 263, 265, 706

— D —

DATA • *100*, *343*, 359, *362*, 366, *372*, 374, 376, 377, 486, 491, *498*, 520, 524, *545*, *546*, *547*, *548*, 590, 689, *695*, 696, 697, 700, 711, 712, 714, 715, 739, *794*, *798*, *799*, *800*, *801*, *807*, *808*, *809*, *828*, *835*, *836*, *837*, *838*, *839*, 840, *841*, *843*, *845*, *846*, 847, *848*, *849*, *852*, *856*, *858*, *861*, *862*, *863*, *865*, *868*, *870*, *872*, *876*, *881*, *883*, *886*, 960, 965
database • 1, 2, 10, 17, 60, 79, 81, 88, 666, 833, 911
data exception • 130, 137, 151, 158, 160, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 188, 192, 193, 194, 198, 229, 230, 245, 262, 264, 283, 289, 291, 317, 318, 319, 320, 321, 322, 525, 570, 573, 634, 636, 646, 684, 696, 700, 904
<data type> • 38, 62, 63, 64, 81, 110, 121, 122, **125**, *126*, 129, 131, 132, *168*, 170, 239, 250, 298, 302, 333, 336, 340, *341*, 342, 345, 366, *372*, 373, 374, 375, 376, 377, 380, 381, *427*, *480*, 481, *486*, 488, *493*, *495*, *539*, *540*, 542, 543, 544, 545, 549, 550, 551, 578, 582, *590*, 677, 680, 681, 684, 705, 750, 752, 923, 927, 932, 937, 946
data type descriptor • 22, 27, 28, 30, 31, 35, 36, 41, 43, 59, 129, 170, 239, 342, 365, 368, 377, 408, 411, 427, 428, 437, 438, 445, 481, 483, 487, 493, 517, 836, 839
<data type list> • *343*, 345, ***480***, 481
DATA_TYPE • 546, 794, 799, 800, 801, 807, 808, 809, 835, 836, 837, 838, 839, 840, 841, 848, 849, 861
DATA_TYPE_DESCRIPTOR • 794, 799, 800, 836, 838, 839, 841, 848, 849
date • 4, 7, 8, 22, 30, 31, 32, 33, 34, 40, 41, 42, 45, 48, 51, 52, 53, 54, 56, 57, 58, 66, 67, 68, 71, 74, 75, 76, 77, 78, 80, 86, 92, 93, 98, 105, 106, 107, 108, 111, 112, 113, 114, 125, 126, 129, 130, 131, 132, 136, 144, 147, 148, 150, 152, 153, 154, 162, 163, 169, 177, 179, 180, 181, 182, 184, 185, 186, 196, 197, 198, 199, 200, 201, 222, 223, 228, 229, 230, 237, 240, 244, 245, 246, 256, 257, 275, 305, 306, 317, 319, 320, 321, 323, 324, 325, 327, 328, 329, 333, 334, 335, 337, 341, 372, 374, 379, 380, 381, 382, 383, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 403, 424, 425, 459, 461, 465, 467, 468, 470, 471, 472, 474, 475, 476, 477, 521, 522, 524, 528, 529, 554, 555, 578, 579, 581, 610, 615, 625, 626, 629, 634, 635, 636, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 664, 688, 690, 697, 700, 702, 703, 704, 705, 711, 713, 718, 719, 720, 730, 731, 733, 740, 741, 754, 783, 789, 839, 857, 904, 905, 916, 921, 922, 923, 924, 925, 928, 930, 931, 932, 933, 938, 947, 948, 951, 957, 960, 963, 969, 970
DATE • 21, 22, 32, *100*, *107*, 111, *126*, 129, 130, 162, 180, 196, 557, 559, 561, 564, 566, 567, 569, 690, 691, 701, *837*
<date literal> • ***107***, 111
<date string> • *98*, ***107***
datetime data type descriptor • 30
<datetime factor> • ***196***
<datetime field> • 31, 32, 33, 34, 111, 113, 129, 130, 131, *152*, 153, 154, 169, 180, 181, 182, 196, 197, 198, 199, 200, 201, 275, 305, 324, **327**, 328, 380, 947
datetime field overflow • 198, 904

<datetime literal> • *106*, **107**, 111, 112, 113, 114, 380, 931
<datetime primary> • ***196***, 197, 198
<datetime term> • ***196***, 197, *199*, 200, 201, 704, 705
<datetime type> • *125*, **126**, 129, 130, 132, 688, 711, 713, 923, 932, 947
<datetime value> • ***108***, 111
<datetime value expression> • *152*, 153, 154, *184*, 185, 186, ***196***, 197, 198, *199*, 200, 201, 933
<datetime value function> • 54, 56, **162**, 163, *196*, *379*, 380, 381, 383, 398, 459, 615, 640, 932
DATETIME_INTERVAL_CODE • *100*, 688, 689, *695*, 697, 700, 701, 711, 713, 714, 715
DATETIME_INTERVAL_PRECISION • *100*, 688, 689, *695*, 697, 700, 701, 711, 713, 714, 715
DATETIME_PRECISION • 794, 799, 836, 837, 838, 839
<date value> • ***107***, 113
DAY • 31, 32, 33, *100*, 111, 130, 197, 324, *327*, 691, 701, 705, *838*
<days value> • ***108***, 111
<day-time interval> • ***108***
<day-time literal> • ***108***, 111
DB-change • 67
DEALLOCATE • *100*, 537, *694*, 706, *708*, 739, 784
<deallocate descriptor statement> • 72, 78, 80, 537, *555*, **694**, 739, 784
<deallocate prepared statement> • 66, 73, 78, 80, 81, *555*, **708**, 739, 927, 929
DEC • *100*, *126*, 127, 576, *774*, 775, *779*
DECIMAL • 21, 22, 28, 40, *100*, *126*, 127, 128, 557, 559, 561, 564, 566, 567, 569, 687, 688, 690, 700, 775, *779*, 780, *837*, 946
DECLARE • *100*, *338*, *490*, *584*, *589*, *604*, *609*, *649*, *721*, 739, *748*, *779*, 780
<declare cursor> • 61, 66, 67, 68, 70, 74, 75, 76, 79, 119, 120, 139, *535*, 536, *584*, 585, 604, **609**, 610, 613, 614, 617, 621, 625, 626, 629, 632, 633, 634, 638, 640, 645, 721, 732, 733, *747*, 749, 751, 753, 921, 937, 950, 956, 959, 961, 963
declared local temporary table • 45, 46, 90, 120, 122, 344, 390, 398, 401, 419, 421, 649, 651, 717, 742, 743, 945
DEFAULT • 23, 35, *100*, *207*, 209, *355*, *379*, *389*, 391, 393, 394, *407*, *432*, 443, 444, *446*, 449, 452, 469, *486*, 488, 520, *531*, *532*, *533*, 571, 626, 628, *631*, 632, 641, 642, *669*, 670, *672*, 674, 785, *793*, *794*, *799*, *815*, *834*, *841*, *848*, 849, *856*, 857, *874*, 875, 933, 938, 951, 961
<default clause> • 41, *372*, 377, **379**, 381, *406*, 408, *427*, 428, *431*, 437, 586, *590*, 959
<default option> • 41, 43, 377, ***379***, 381, 383, 841, 849, 918, 935
<default specification> • ***207***, 208
default SQL-connection • 89, 670, 672, 674
default SQL-session • 89, 571, 670, 785
default time zone • 91, 111, 113, 130, 147, 945
DEFAULT_CHARACTER_SET_CATALOG • 793, 834
DEFAULT_CHARACTER_SET_NAME • 793, 834
DEFAULT_CHARACTER_SET_SCHEMA • 793, 834
DEFAULT_COLLATE_CATALOG • 815, 874, 875
DEFAULT_COLLATE_NAME • 815, 874, 875, 961
DEFAULT_COLLATE_SCHEMA • 815, 874, 875
DEFERRABLE • *100*, *352*, 353, 384, 427, 459, 657, *796*, *810*, *814*, 842, *843*, 844, *852*, 853, *862*, 925
DEFERRED • *100*, *352*, 615, *657*, *796*, *810*, *814*, 842, *843*, 844, *852*, 853, *862*
<deferred assertion trigger> • ***459***, 461
Definition Schema • 92, 94, 104, 118, 789, 830, 863
DEFINITION_SCHEMA • 92, 120, 789, 793, 794, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 830, 831, 890, 891, 892, 893, 894, 895, 896, 897
degree • 36, 44, 47, 48, 86, 144, 208, 209, 210, 220, 227, 229, 238, 251, 261, 268, 273, 278, 296, 298, 302, 305, 310, 367, 369, 376, 377, 404, 411, 421, 610, 617, 618, 633, 641, 645, 652, 710, 714, 726, 787, 899, 900
DELETE • 58, 82, 83, *100*, *237*, *343*, *362*, 366, 370, *389*, 390, 391, 398, 423, *459*, 461, *465*, 467, 469, 470, 471, 472, 474, 476, 477, 504, 521, *625*, 626, *628*, *649*, 650, 651, 653, 664, *729*, *732*, 739, 740, *811*, *856*, 857, *863*, 864, *868*, 869, *881*, 882
<delete rule> • 228, ***389***, 391, 393, 396, 857
<delete statement: positioned> • 67, 71, 74, 75, 77, 78, 244, 257, 336, 392, 396, 461, *554*, **625**, 626, 627, 629, 640, 641, 645, 729, 732, 739, 754
<delete statement: searched> • 67, 71, 74, 76, 78, 144, 147, *237*, 244, 245, 257, 336, 395, 461, *465*, 467, 468, 470, 471, 472, 477, 524, *554*, 596, 626, **628**, 629, 630, 641, 664, *702*, 739, 741, *783*, 960
DELETE_RULE • 811, 856, 857
<delimited identifier> • 92, ***98***, 103, 104, *115*, 789
<delimited identifier body> • ***98***, 104, 105, 118, 789, 922
<delimited identifier part> • ***98***, 104
<delimiter token> • ***98***, 103, 922, 931
depend • 4, 9, 13, 14, 15, 17, 21, 36, 40, 42, 43, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 62, 63, 64, 65, 67, 68, 77, 82, 83, 84, 87, 89, 90, 91, 111, 112, 146, 162, 174, 201, 209, 219, 225, 238, 239, 242, 250, 251, 252, 274, 275, 315, 323, 338, 339, 342, 364, 370, 384, 394, 399, 409, 410, 417, 421, 426, 427, 430, 434, 437, 450, 514, 515, 516, 519, 523, 525, 529, 536, 537, 543, 551, 570, 577, 578, 585, 586, 591, 604, 612, 613, 615, 619, 623, 624, 626, 629, 634, 640, 645, 650, 652, 656, 661, 674, 696, 697, 700, 701, 707, 711, 712, 713, 741, 751, 752, 753, 757, 784, 787, 788, 808, 809, 818, 819, 822, 826, 874, 905, 951, 955, 956, 957, 958, 959, 960, 961

dependent • 4, 9, 13, 14, 15, 17, 21, 36, 42, 43, 46, 47, 48, 49, 51, 52, 64, 65, 67, 68, 82, 83, 84, 87, 89, 90, 91, 111, 112, 146, 162, 174, 201, 209, 219, 225, 238, 239, 242, 250, 251, 252, 274, 275, 323, 338, 339, 342, 364, 370, 384, 394, 409, 410, 417, 421, 426, 427, 430, 434, 437, 450, 514, 515, 519, 523, 525, 529, 536, 537, 543, 551, 570, 577, 578, 585, 586, 591, 604, 612, 613, 619, 623, 624, 626, 629, 634, 640, 645, 650, 652, 656, 661, 674, 696, 697, 700, 701, 707, 711, 712, 713, 741, 751, 752, 753, 757, 784, 787, 788, 808, 809, 818, 819, 822, 826, 874, 905, 951, 955, 956, 957, 958, 959, 960, 961
dependent privilege descriptors still exist • 519, 905
deprecated • 64, 92, 93, 261, 544, 754, 762, 765, 769, 772, 775, 777, 781, 910
DEPTH • *100*, *260*, 262, 263, 264, 965
DEREF • 40, 41, *100*, *164*, 966
<deref function> • ***164***
<derivation clause> • ***372***
<derivation functions> • ***372***
<derived column> • 147, 222, ***237***, 238, 239, 246, 611, 619, 711, 916
<derived column list> • ***143***, 144, 145, 932
<derived table> • ***143***, 144, 145, 219, 240, 252, 254, 307, 466, 923, 924
DESC • *100*, *446*, 449, *609*, 612
DESCRIBE • *100*, *709*, 739
<describe input statement> • 73, 78, 80, ***709***, 712, 714, 927, 929, 960
<describe output statement> • 73, 78, 80, ***709***, 710, 715, 960
<describe statement> • *555*, 701, **709**, 739, 952
descriptor • 8, 13, 17, 22, 23, 24, 27, 28, 30, 31, 35, 36, 37, 38, 41, 43, 44, 47, 48, 49, 50, 51, 58, 59, 60, 61, 62, 72, 73, 78, 80, 81, 82, 83, 84, 85, 91, 117, 121, 123, 124, 129, 131, 141, 170, 208, 217, 219, 222, 223, 225, 226, 233, 239, 242, 248, 249, 250, 251, 252, 259, 322, 342, 344, 346, 348, 349, 356, 359, 363, 365, 366, 367, 368, 369, 370, 376, 377, 379, 381, 382, 383, 384, 385, 387, 390, 391, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 413, 415, 416, 417, 419, 420, 421, 422, 423, 424, 426, 427, 428, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 441, 443, 444, 445, 447, 448, 450, 452, 455, 456, 457, 459, 460, 463, 469, 473, 478, 481, 482, 483, 484, 487, 491, 492, 493, 494, 496, 499, 500, 502, 503, 504, 505, 507, 508, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 523, 524, 529, 537, 543, 551, 552, 555, 611, 615, 624, 629, 632, 639, 644, 650, 651, 653, 681, 683, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 726, 738, 739, 740, 784, 836, 839, 841, 843, 845, 846, 848, 849, 852, 861, 862, 863, 865, 868, 870, 872, 874, 876, 878, 881, 891, 892, 905, 906, 907, 908, 929, 952, 955, 960
DESCRIPTOR • *100*, 537, *546*, *547*, *548*, *692*, *694*, *695*, *699*, *710*, 738, 739, 740, 784, *794*, *799*, *800*, *836*, *838*, *839*, *841*, *848*, *849*, *863*, *865*, *890*, *891*, *892*, *893*, *894*, *895*, *896*
<descriptor item name> • ***695***, 696, *699*, 700
<descriptor name> • 80, **117**, 121, 123, 124, 523, 529, 537, *692*, 693, *694*, *695*, 696, 697, 698, *699*, 700, 701, *710*, 712, 713, 715, 716, 784, 929, 952
designated columns • 387
DESTROY • *100*, *491*, *596*, 739
<destroy statement> • 72, 541, *554*, **596**, 629, 739, 921
DESTRUCTOR • *100*, 336, *491*, *539*, 541, 596
determinant • 51, 52, 54
determines • 24, 25, 28, 29, 51, 83, 89, 492, 944
deterministic • 49, 235, 241, 252, 398, 422, 460
DIAGNOSTICS • *100*, *604*, *605*, *655*, *735*
diagnostics area • 49, 58, 65, 85, 91, 477, 555, 571, 572, 578, 579, 656, 735, 738, 741, 784, 786, 903, 952, 956, 957
diagnostics area limit • 85, 91, 656
<diagnostics size> • ***655***
DICTIONARY • *100*, *446*, *816*, *876*, 877, 966
<dictionary name> • ***446***
<digit> • ***95***, *98*, *107*, 110, 113, *115*, 176, 178, 293, 295, *755*, 903, 923, 953
<direct implementation-defined statement> • ***783***, 784, 785, 953

directly • 14, 38, 45, 53, 54, 55, 56, 60, 66, 69, 76, 79, 81, 88, 90, 91, 110, 118, 120, 146, 149, 185, 198, 222, 231, 232, 235, 236, 238, 241, 242, 244, 256, 272, 315, 376, 427, 443, 448, 488, 493, 495, 514, 515, 516, 519, 536, 552, 597, 599, 600, 611, 639, 643, 677, 678, 680, 730, 741, 783, 893, 894, 895, 896, 933, 960
directly based on • 488, 493, 495
directly contain • 14, 53, 54, 55, 56, 146, 149, 185, 198, 222, 231, 232, 235, 236, 238, 241, 244, 256, 315, 376, 427, 443, 448, 536, 552, 597, 599, 600, 611, 639, 643, 741, 933
<directly executable statement> • ***783***
<direct select statement: multiple rows> • 71, 74, 75, 76, 78, 144, 147, 739, *783*, 784, **787**
<direct SQL data statement> • ***783***, 786, 931, 941
<direct SQL statement> • 82, 85, 89, 90, 91, 110, 118, 120, 572, 677, 678, 680, **783**, 784, 785, 786, 887, 911, 913, 931, 940, 945
direct subtable • 46, 47, 370, 413, 415, 642, 647
direct supertable • 47, 48, 367, 368, 370, 413
DISCONNECT • *100*, *548*, *674*, 739
disconnect error • 675, 908
<disconnect object> • ***674***
<disconnect statement> • 72, 89, *554*, **674**, 675, 739, 928, 960
disjoint • 52, 615, 644
DISTINCT • *100*, *149*, 151, 241, 242, 243, 244, 246, 275, *310*, 311, 492, *493*, 525, 611, *794*, *796*, *799*, *802*, *803*, *899*, 923, 924, 958
<distinct predicate> • *271*, 272, **310**, 312, 934
<distinct type definition> • *483*, **493**
<distinct type name> • ***493***
division by zero • 188, 904
DO • *101*, *338*, *604*
DOMAIN • *101*, *343*, 358, *427*, *430*, *436*, 438, *828*, *829*
<domain constraint> • 41, 135, 136, 183, ***427***, 428, *433*, 828
domain constraint descriptor • 41, 49, 51, 408, 409, 428, 434, 436, 437, 517, 519, 843
<domain definition> • 69, 110, 127, 128, 136, *355*, 356, 357, 379, 398, 399, 409, **427**, 428, 429, 437, 514, *553*, 739, 935, 936
domain descriptor • 41, 59, 91, 322, 381, 382, 383, 408, 409, 427, 428, 430, 431, 432, 433, 434, 436, 437, 452, 481, 504, 514, 517, 518, 519, 520, 841
<domain name> • 41, *107*, 110, **115**, 122, 124, *168*, 170, 171, 183, *194*, 195, 239, *343*, 344, 358, 372, 373, 374, 375, 376, 377, 408, *427*, 428, *430*, 431, 432, 433, 434, *436*, 483, 504, 512, 514, 519, *590*, 705, 849, 931, 935
DOMAINS • 794, 809, 822, 839, 841, 843, 848, 872, 898, 931
DOMAIN_CATALOG • 794, 796, 799, 800, 809, 822, 836, 838, 839, 841, 843, 848, 849, 861, 872
DOMAIN_DEFAULT • 794, 841
DOMAIN_NAME • 794, 796, 799, 800, 809, 822, 836, 838, 839, 841, 843, 848, 849, 861, 872
DOMAIN_SCHEMA • 794, 796, 799, 800, 809, 822, 836, 838, 839, 841, 843, 848, 849, 861, 872
dormant SQL-connection • 88, 89, 670
dormant SQL-session • 89, 670, 672
DOUBLE • 21, *22*, 28, 40, *101*, *126*, 128, *545*, 557, 559, 561, 564, 566, 567, 569, 687, 688, 690, 701, *760*, 761, 765, *771*, 772, *836*, 946, 947
<double colon> • *98*, **99**, *107*
<double period> • *98*, **99**, *760*, *776*
<double quote> • *95*, **96**, *98*, 103, 104
<doublequote symbol> • ***98***, 103, 104
DROP • *101*, *358*, 359, *407*, *408*, *410*, 411, *415*, *417*, *419*, 420, *426*, *432*, *434*, *436*, 437, *441*, *445*, *452*, *457*, *463*, *478*, *480*, *497*, *498*, 499, *502*, *511*, 519, 520, 651, 653, 739
<drop assertion statement> • 69, 358, **463**, 519, *554*, 739, 927, 929, 931
<drop behavior> • ***358***, *410*, *415*, *417*, *419*, *426*, *436*, *452*, 453, *480*, 497, *498*, *502*, *512*, 919
<drop character set statement> • 70, 359, **445**, *553*, 739, 937
<drop collation statement> • 70, 359, **452**, 453, *553*, 739, 926, 927, 929, 931
<drop column default clause> • *405*, **407**, 936
<drop column definition> • *401*, **410**, 412, 415, 936
<drop column domain clause> • *405*, **408**, 409
<drop data type statement> • **498**, 499, 520, *554*, 556, 739, 835, 919, 920
<drop domain constraint definition> • *430*, **434**, 435, 926
<drop domain default clause> • *430*, **432**, 926
<drop domain statement> • 69, 358, **436**, 438, 519, *553*, 739, 936
<drop module statement> • 61, 70, 359, 499, **502**, *554*, 556, 739, 919, 920
<drop null class statement> • 70, 359, **441**, *553*, 556, 866, 867, 918, 920
<drop role statement> • 70, **511**, *553*, 556, 739, 833, 920
<drop routine statement> • 70, 359, **480**, 482, *554*, 556, 919, 920
<drop schema statement> • 69, **358**, 360, *553*, 739, 935
<drop supertable clause> • *401*, 402, **415**, 918
<drop table constraint definition> • *401*, 415, **417**, 418, 936
<drop table statement> • 69, 358, **419**, 420, 498, *553*, 651, 739, 935
<drop translation statement> • 70, 359, **457**, *554*, 739, 926, 927, 929, 931
<drop trigger statement> • 70, 359, 410, **478**, 519, *554*, 556, 739, 919, 920
<drop type template statement> • 70, 359, **497**, 520, *554*, 556, 920
<drop view statement> • 69, 358, 411, **426**, 498, 519, *553*, 739, 936
duplicate asynchronous SQL statement identifier • 571, 906
dyadic • 8, 25, 26, 187, 188, 191, 205, 703

dyadic operator • 8, 25, 26, 703
<dynamic close statement> • 66, 71, 73, 75, 76, 77, 80, *555*, 578, **728**, 739, 754
<dynamic cursor name> • **117**, 123, 124, *724*, *726*, *728*, *729*, *730*, 731, 930, 931, 952
<dynamic declare cursor> • 66, 70, 74, 75, 76, 79, 80, *535*, 536, 538, 707, **721**, 724, 726, 728, 729, 730, 732, 733, *747*, 750, 751, 753, 929, 937, 940
<dynamic delete statement: positioned> • 67, 71, 74, 75, 77, 78, 80, *555*, 578, **729**, 739, 754
<dynamic fetch statement> • 67, 71, 73, 75, 76, 77, 80, *555*, 578, 618, 701, 710, 714, 715, **726**, 739, 754, 930, 952
<dynamic open statement> • 66, 67, 71, 73, 75, 76, 77, 80, *555*, 578, 701, 713, 721, 722, **724**, 739, 754, 952
<dynamic parameter specification> • 81, *133*, ***134***, 135, 136, 137, 138, 199, 468, 696, 697, 700, 703, 709, 712, 713, 714, 718, 719, 724, 783, 932
<dynamic select statement> • 71, 74, 75, 76, 77, 79, 80, 578, ***702***, 710, 713, 714, 717, 719
<dynamic single row select statement> • 71, 74, 75, 76, 77, 78, 80, 148, 578, ***702***, 710, 714, 717, 718, 719, 740, 929
dynamic SQL error • 692, 696, 699, 707, 713, 714, 715, 717, 718, 722, 724, 906
<dynamic update statement: positioned> • 67, 71, 74, 75, 77, 78, 80, 424, *555*, 578, **730**, 731, 740, 754
DYNAMIC_FUNCTION • *100*, 606, *735*, 737, 738, 744

— E —

EACH • *101*, *459*, 461, *465*, 468, 469, 475, 884, 966
Editor's Note • 12, 34, 40, 41, 57, 60, 66, 68, 81, 93, 203, 223, 239, 242, 272, 288, 359, 366, 369, 387, 402, 424, 437, 438, 452, 466, 473, 487, 517, 543, 573, 574, 575, 577, 615, 625, 629, 634, 642, 645, 647, 667, 687, 717, 754, 789, 801, 863, 897
effective • 15, 16, 21, 25, 27, 32, 33, 34, 40, 42, 45, 46, 49, 58, 66, 68, 86, 89, 90, 91, 105, 111, 112, 113, 130, 144, 148, 162, 197, 201, 227, 228, 229, 232, 236, 245, 250, 254, 258, 274, 275, 276, 310, 311, 322, 335, 336, 337, 338, 342, 348, 353, 356, 358, 359, 364, 365, 366, 374, 387, 391, 395, 396, 399, 404, 409, 410, 411, 413, 415, 417, 420, 425, 426, 428, 434, 437, 438, 440, 441, 443, 445, 448, 449, 450, 452, 455, 457, 460, 466, 474, 475, 477, 481, 487, 488, 497, 498, 499, 502, 504, 505, 511, 519, 520, 537, 550, 571, 572, 590, 612, 614, 615, 622, 626, 627, 629, 634, 635, 636, 641, 645, 646, 649, 650, 651, 653, 664, 678, 700, 714, 715, 741, 750, 784, 785, 788, 836, 841, 843, 845, 846, 848, 852, 861, 862, 868, 870, 872, 881, 945, 950, 956, 957
ELEMENT • *101*, *631*, 966
ELSE • *101*, *165*, 166, *599*
<else clause> • ***165***, 166
ELSEIF • *101*, *599*, 966
embedded • 1, 2, 66, 68, 69, 73, 74, 75, 76, 78, 79, 80, 81, 92, 95, 133, 134, 135, 137, 139, 140, 317, 524, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 767, 768, 769, 771, 772, 774, 775, 776, 777, 779, 780, 781, 783, 887, 911, 913, 922, 930, 940, 944, 949, 950, 952, 961, 963
<embedded character set declaration> • *747*, ***748***, 749, 751, 753, 754, 940, 952
<embedded exception declaration> • 73, *747*, 753, **755**, 756, 757, 758, 759, 963
<embedded SQL Ada program> • 79, *747*, 749, 751, 753, 756, **760**, 761, 762
<embedded SQL begin declare> • *747*, **748**, 749, 753, 776
<embedded SQL COBOL program> • 79, *747*, 749, 751, 753, 756, **767**, 768, 769, 963
<embedded SQL C program> • 79, *747*, 749, 751, 753, 756, **763**, 764, 765
<embedded SQL declare section> • 79, **747**, 748, 749, 753, 754, 761, 764, 768, 771, 774, 776, 779
<embedded SQL end declare> • *747*, **748**, 749, 753, 776
<embedded SQL Fortran program> • 79, *747*, 749, 751, 753, 754, 757, **771**, 772
<embedded SQL host program> • 79, 81, **747**, 748, 749, 750, 751, 754, 756, 757, 887, 911, 913, 961
<embedded SQL MUMPS declare> • *747*, **748**, 749, 753
<embedded SQL MUMPS program> • 79, *747*, 748, 749, 751, 757, **774**, 775
<embedded SQL Pascal program> • 79, *747*, 749, 751, 753, 757, **776**, 777
<embedded SQL PL/I program> • 79, *747*, 749, 751, 753, 757, **779**, 780, 781
<embedded SQL statement> • 79, 135, 139, **747**, 748, 749, 750, 753, 754, 761, 764, 768, 771, 774, 776, 779, 922, 930, 940
<embedded variable name> • *133*, *134*, 135, 137, *139*, 140, 524, **748**, 750, 751, 752, 961
<embedded variable specification> • 135, ***139***, 140, 317, 783
ENABLED_APPLICABLE_ROLES • 890, 891, 892, 897
ENABLED_ROLE • 889, 890, 897
ENABLED_ROLES • 889, 890, 897
<encapsulation level> • 367, 368, ***372***, 373, 374, 376, 377, 403, *486*, 624, 632, 917
encompassing transaction • 88, 664, 666
END • *101*, *165*, 166, *338*, 376, *500*, *535*, *584*, *597*, *599*, *602*, *604*, *748*, *798*, *799*
END-EXEC • *101*, *747*, 749
<end field> • 201, 324, ***327***, 328, 329

<ending label> • ***584***, *602*, *604*
<enumerated collating sequence> • ***447***, 449
<enumerated primary> • ***194***
<enumerated type> • 110, 122, *125*, **126**, 128, 131, 274, 380
<enumerated value expression> • *184*, 185, 186, **194**, 195, 915, 916
<enumeration literal> • *106*, **107**, 110, 112, 113, 195, 380, 549, 915
<enumeration name> • *107*, 110, **115**, 122, *126*, 128, 177, 179, 182
<enumeration name list> • ***126***
environment • 2, 17, 18, 59, 60, 88, 89, 91, 93, 346, 507, 536, 537, 670, 944, 950, 951, 957
EQUALS • *101*, *485*, *488*, *531*, *532*, *533*, *801*, *835*, 966
<equals clause> • 274, *363*, 371, 386, ***485***, 488, 492, 917
<equals function specification> • *485*, **486**, 488, 489
<equals operator> • ***96***, *273*, *447*, 449, *455*, *592*, *638*, *695*, *699*, *735*, *760*, *764*
EQUAL_KEY_DEGREES • 899
equivalent • 1, 16, 17, 37, 54, 79, 80, 88, 104, 109, 110, 118, 122, 127, 146, 148, 165, 166, 174, 175, 208, 225, 238, 248, 251, 278, 280, 282, 289, 293, 313, 328, 338, 344, 375, 376, 408, 436, 448, 512, 599, 604, 611, 632, 650, 750, 754, 761, 762, 765, 768, 769, 772, 774, 775, 777, 780, 781, 789, 969
error in assignment • 696, 700, 904
ESCAPE • *101*, 159, *282*, 283, *286*, 291, *546*
<escape character> • *155*, 156, 158, ***282***, 283, 285, *286*, 287, 288, 289, 290, 706, 934
<escaped character> • ***287***, 288, 289
<exact numeric literal> • ***107***, 110, 112, 175, 178, 380
<exact numeric type> • 29, *125*, **126**, 128, 548, 711, 713, 762, 946, 949
EXCEPT • *101*, *247*, 249, 255, 257, 258, 628, 632, *845*, 933
exception • 13, 14, 15, 40, 42, 49, 58, 64, 65, 69, 72, 73, 74, 75, 76, 78, 79, 81, 85, 87, 89, 103, 111, 117, 122, 130, 137, 151, 158, 160, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 188, 192, 193, 194, 198, 229, 230, 245, 262, 263, 264, 266, 268, 283, 289, 291, 317, 318, 319, 320, 321, 322, 338, 339, 340, 353, 393, 394, 395, 396, 424, 477, 519, 524, 525, 529, 535, 536, 538, 550, 555, 570, 571, 572, 573, 577, 578, 584, 585, 586, 587, 588, 589, 596, 598, 602, 606, 607, 614, 615, 616, 618, 619, 621, 623, 624, 626, 628, 629, 633, 634, 636, 640, 641, 644, 645, 646, 656, 657, 659, 660, 661, 663, 664, 665, 666, 667, 669, 670, 672, 674, 677, 678, 680, 681, 683, 684, 692, 694, 696, 699, 700, 701, 703, 706, 707, 708, 709, 710, 713, 714, 715, 717, 718, 719, 722, 724, 735, 738, 741, 742, 744, 747, 753, 754, 755, 756, 757, 758, 759, 784, 785, 786, 903, 904, 906, 907, 908, 910, 920, 944, 953, 955, 960, 963
EXCEPTION • *101*, *102*, *260*, 262, 263, 266, *545*, *546*, *547*, 585, *587*, *589*, *604*, *605*, 606, 607, *735*, *755*, 756, 758, 759, 922, 967
<exception declaration> • 72, *535*, 536, 538, **589**, 606, 607, 920
<exception handler> • 72, *584*, 585, 586, **587**, 588, 920
<exception handler action> • ***587***, 588, 607
<exception handler condition> • 536, ***587***
<exception handler contents> • ***587***, 588
<exception name> • **117**, 122, 536, *587*, *589*, *606*, *607*
<exception name list> • ***587***
<exception value> • 122, 587, ***589***
EXEC • *101*, *747*, 749
EXECUTE • 82, 84, *101*, 337, *343*, 344, 347, 500, 502, *546*, 581, *717*, *719*, 740, 917
<execute immediate statement> • 16, 59, 72, 77, 80, 81, 90, 110, 118, 120, 135, *555*, 677, 678, 680, **719**, 740, 944, 945
<execute statement> • 16, 73, 77, 80, 81, *555*, 701, 706, 707, 713, 714, 715, **717**, 718, 740, 952
<existential clause> • 147, ***307***, 308, 525
<existing character set name> • ***442***, 443
<existing type name> • ***362***
EXISTS • *101*, *300*, 385, 387, 428, *799*, *845*, *899*, *900*
<exists predicate> • 238, *271*, **300**, 524
explicit • 11, 13, 17, 23, 25, 32, 41, 49, 58, 59, 67, 85, 86, 87, 88, 90, 91, 103, 120, 121, 124, 130, 149, 154, 166, 180, 181, 208, 240, 248, 249, 251, 258, 329, 339, 345, 353, 356, 358, 363, 368, 369, 384, 387, 395, 396, 401, 403, 408, 410, 419, 421, 422, 426, 427, 428, 430, 436, 437, 439, 443, 445, 447, 448, 449, 452, 455, 457, 459, 460, 463, 469, 470, 471, 472, 473, 480, 481, 483, 488, 493, 494, 496, 497, 498, 500, 503, 512, 513, 514, 515, 516, 531, 532, 533, 536, 537, 541, 542, 543, 544, 549, 551, 570, 585, 615, 617, 619, 625, 627, 639, 656, 705, 706, 726, 730, 744, 751, 846, 851, 858, 882, 923, 924, 944, 945, 948, 949, 951, 955, 957
Explicit • 25, 26, 27, 40, 41, 191, 233, 611
<explicit abstract data type definition> • ***483***, 541
<explicit table> • ***248***, 249, 258, 924, 957
<exponent> • ***107***, 112
exposed • 143, 145, 240, 245, 307
<extended cursor name> • 80, ***117***, 121, 123, 124, 703, 719, *722*, 724, 923
<extended statement name> • 80, ***117***, 121, 123, 124, 706, 707, *722*, 923, 960
EXTERNAL • 82, 84, *101*, *343*, 344, 345, 359, *446*, *454*, 498, 503, 516, 517, *540*, *547*, *548*, 551, 739, 949
<external body reference> • 62, ***540***
<external collation> • ***446***, 448, 450
<external collation name> • ***446***, 448
external function exception • 339, 906

external function invocation exception • 340
<external parameter> • ***539***, 549, 550
<external routine language clause> • 339, ***540***, 544, 549, 551, 552, 959
<external routine name> • **117**, 123, 339, *540*, 549, 552
<external translation> • ***454***
<external translation name> • ***454***, 455
EXTRACT • *101*, *152*, 154, 703
<extract expression> • 30, 34, ***152***, 153, 154, 932, 947
<extract field> • ***152***, 153, 154, 703
<extract source> • ***152***, 153, 154

— F —

<factor> • ***187***, *199*, 200
FALSE • *101*, *108*, 177, 179, *313*
feature not supported • 669, 672, 906
FETCH • *101*, *546*, *604*, *605*, *617*, *726*, 739, 740
<fetch orientation> • ***617***, 618, 619, 620, *726*, 927, 930
<fetch statement> • 67, 71, 73, 75, 76, 77, *554*, 578, **617**, 618, 620, 621, 740, 754, 927, 937, 959
<fetch target list> • ***617***, 619, 620, 726, 937, 959
FIRST • *101*, *260*, *545*, *617*, 618, 619
fixed-length coding • 7, 9
FLOAT • 21, 22, 28, 40, *101*, *126*, 128, 557, 559, 561, 564, 566, 567, 569, 687, 688, 690, 701, *779*, 781, *836*, 946, 947
<fold> • 24, ***155***, 156, 159, 161, 704, 923
FOR • *101*, *155*, 177, 179, *307*, *338*, *446*, *454*, *459*, 460, 461, *465*, 468, 469, 475, *512*, 515, 516, 517, 518, 519, 573, 574, 575, 576, *589*, *604*, *609*, 610, 611, 613, 615, 639, 704, *721*, *722*, 730, 919, 927, 930
FOREIGN • *101*, *221*, 224, 225, 226, 228, 375, *389*, 626, 628, 633, 639, 644, *801*, *834*, *835*, *839*, *841*, *843*, *845*, *848*, *850*, *851*, *852*, 853, *854*, *856*, *859*, *860*, *861*, *862*, *863*, *865*, *866*, *867*, *868*, *870*, *872*, *874*, *876*, *878*, *880*, *881*, *883*, *885*, *889*, *900*, *901*
<for loop name> • ***604***
<for loop specification> • ***604***
form-of-use • 9, 22, 23, 24, 117, 122, 123, 124, 155, 156, 157, 159, 160, 161, 176, 177, 179, 348, 349, 442, 704, 874, 903, 923, 946, 948, 949
form-of-use conversion • 9, 22, 24, 122, 123, 124, 156, 159, 160, 161, 704, 923, 946, 948
<form-of-use conversion> • 24, ***155***, 156, 159, 160, 161, 704, 923, 948
<form-of-use conversion name> • **117**, 122, 123, 124, *155*, 156, 160, 923, 946
<form-of-use specification> • ***442***
FORM_OF_USE • 815, 874, 875, 953
<for statement> • 72, *554*, 585, **604**, 605, 921, 959
FORTRAN • 3, *100*, *331*, 332, 339, 544, 550, 574, 751, 835, *886*, 887, 888
<Fortran host identifier> • *748*, ***771***, 772
<Fortran type specification> • ***771***, 772, 773, 930, 940
<Fortran variable definition> • *748*, **771**, 772
FOUND • *101*, *447*, 448, 449, *545*, *755*, 756, 758
FROM • 54, 55, 56, *101*, *152*, 154, *155*, 157, 172, 177, 179, 180, 181, 182, *219*, 221, 223, 225, 227, 249, 251, *310*, 311, 385, 386, 387, 391, 411, 420, 426, 428, 437, 438, 441, *442*, 443, 445, *446*, 452, *454*, 457, 481, 497, 499, 502, *510*, 511, *512*, *540*, 573, 574, 575, 576, *617*, 620, *625*, *628*, 664, 669, 670, 677, 678, 680, 681, 692, *702*, 703, 704, 706, 722, *726*, *729*, *732*, 741, *791*, *792*, *793*, *794*, *796*, *797*, *798*, *799*, *800*, *801*, *802*, *803*, *804*, *805*, *806*, *807*, *808*, *809*, *810*, *811*, *812*, *813*, *814*, *815*, *816*, *817*, *818*, *819*, *820*, *821*, *822*, *823*, *824*, *825*, *826*, *827*, *832*, *833*, *839*, *841*, *845*, *846*, *848*, *849*, *850*, *851*, *852*, *853*, *854*, *856*, *858*, *859*, *860*, *863*, *865*, *872*, *874*, *876*, *878*, *890*, *891*, *892*, *893*, *894*, *895*, *896*, *897*, *898*, *899*, *900*, 937
<from clause> • 54, 143, *217*, 218, **219**, 220, 231, 233, 235, 240, 241, 307, 308, 309, 524, 626, 632, 924, 928, 930
FULL • 50, 54, *101*, *221*, 223, 227, *302*, 303, *389*, 391, 392, 393, 394, 633, 640, 644, *856*, 857, *886*, *900*
FUNCTION • *99*, *100*, *101*, 359, 376, 481, *488*, *490*, *491*, *522*, *523*, *526*, *528*, *529*, *531*, *532*, *533*, *539*, 540, *547*, *548*, 606, *735*, 737, 738, 739, 744
functional dependencies • 48, 52, 53, 54, 55, 56, 57
functionally dependent • 51, 52, 238
<function invocation> • ***133***, 136, 171, *237*, 244, 245, 257, *379*, 381, 383, 918
<function type> • ***539***, 541, 552

— G —

GENERAL • *101*, 124, 966
<generalized expression> • ***333***
<general literal> • ***106***, 113, 114, 915
generally contain • 14, 151, 232, 398, 421, 422, 427, 433, 459, 460, 466, 467, 472, 615, 630, 637, 639, 647, 730, 928, 932
general null • 41, 42, 111, 135, 136, 137, 151, 153, 157, 158, 159, 160, 162, 166, 171, 173, 185, 187, 188, 192, 194, 197, 200, 203, 205, 206, 209, 211, 213, 215, 227, 229, 273, 274, 278, 282, 283, 284, 289, 291, 297, 299, 317, 322, 324, 340, 381, 382, 393, 394, 440, 641, 645, 711, 714, 840, 956
<general set function> • ***149***, 150, 151, 923, 932
<general value specification> • 54, 56, ***133***, 137, 138, 428, 572, 932, 947
generated type • 9, 10, 37, 38, 131, 341, 342, 367, 369, 487, 495, 523, 526, 529, 531, 532, 533
<generated type reference> • 38, *125*, 131, **341**, 342, 367, 369, 495
GET • *101*, *372*, *442*, *695*, *735*, 740
<get count> • ***695***
<get descriptor information> • ***695***
<get descriptor statement> • 73, 78, 80, *555*, **695**, 696, 740, 960

<get diagnostics statement> • 73, *555*, **735**, 737, 738, 741, 745, 940
get function • 8, 43, 142, 185, 234, 374, 377
<get function> • ***372***, 374
<get item information> • ***695***
<get key type> • ***695***
GLOBAL • 45, *101*, *117*, 123, *361*, 369, *845*, 925
<global or local> • ***361***
global temporary table • 45, 46, 47, 90, 358, 361, 369, 390, 398, 925, 945, 956
GO • *101*, *755*, 756, 757, 758, 759
GOTO • *101*, *755*
<go to> • ***755***, 756, 757, 758, 759
<goto target> • ***755***, 756, 757
GRANT • 83, 91, *101*, 348, 443, 448, 455, *503*, 504, 505, *508*, *512*, 514, 515, 516, 517, 518, 519, 789, 864, 869, 871, 873
GRANTEE • 438, 441, 481, 499, 794, 797, 800, 801, 802, 803, 804, 805, 806, 807, 815, 816, 817, 861, 863, 864, 865, 868, 870, 872, 873, 890, 891, 892, 893, 894, 895, 896
<grantee> • **344**, 346, *503*, 504, 505, *508*, *510*, *512*, 520, 861, 864, 865
GRANTOR • 804, 805, 806, 807, 861, 863, 868, 870, 872, 873, 891, 892
<grant role statement> • 70, 82, *355*, 357, **508**, *553*, 556, 740, 865, 917, 919, 920
<grant statement> • 70, 83, 84, 344, 348, *355*, 443, 448, 455, **503**, 504, 505, *553*, 740, 861, 863, 864
<greater than operator> • ***96***, *273*, *447*, 449, 612, 788
<greater than or equals operator> • *98*, **99**, *273*
GROUP • *101*, *233*, 239, *801*, *898*, *899*
<group by clause> • 45, 55, 143, *217*, 218, **233**, 235, 241, 269, 421, 492, 624, 924, 927, 934
grouped table • 45, 149, 217, 233, 235, 238, 242, 243, 611
grouped view • 218, 220, 246, 269, 421, 624, 924, 927, 934
<grouping column reference> • ***233***
<grouping column reference list> • ***233***, 238

— H —

HAVING • *101*, *235*
<having clause> • 45, 55, 143, 149, *217*, 218, 222, 231, **235**, 241, 269, 421, 492, 624, 924, 927, 934
<hexit> • *106*, ***107***, 109, 110
<hex string literal> • *98*, 103, ***106***, 109, 110, 112, 113, 380, 922, 923
<high> • ***18***, 19
<host identifier> • ***748***, 750, 753
<host label identifier> • ***755***, 756, 757, 758, 759
<host PL/I label variable> • ***755***, 756, 757, 758, 759
<host variable definition> • *747*, ***748***, 750, 751, 752, 753, 754
HOUR • 31, *32*, 33, 90, *101*, *102*, 111, 130, 131, 148, *152*, 154, 196, 197, 324, *327*, 684, 691, 701, *838*
<hours value> • *107*, ***108***, 111

— I —

<identifier> • 59, 61, 80, 88, 90, ***115***, *116*, *117*, 118, 121, 124, *287*, 288, *439*, *442*, *465*, 468, 543, 544, *584*, *604*, 661, 667, 670, 680, 692, 706, 710, 722, 737, 744, 789, 828, 923, 946
<identifier body> • ***98***, 104, 105, 118, 789, 922, 931
<identifier part> • ***98***, 104
<identifier start> • ***98***, 103, 104
IDENTITY • *101*, *454*, 456
IF • *101*, *599*, 966
<if statement> • 72, *554*, **599**, 600, 921
<if statement else clause> • ***599***, 600
<if statement elseif clause> • ***599***
<if statement then clause> • ***599***, 600
IGNORE • *101*, *447*, 449, 966
IMMEDIATE • *101*, *352*, 353, 384, 427, 459, 615, 622, *657*, 664, *719*, 740, *862*, 925
<immediate assertion trigger> • ***459***, 460, 461
immediate base • 424
immediate constituent • 264
immediately contain • 14, 16, 112, 113, 143, 144, 147, 148, 153, 185, 187, 196, 197, 198, 203, 204, 207, 208, 211, 212, 213, 214, 215, 231, 238, 241, 249, 252, 268, 307, 308, 335, 366, 375, 385, 387, 422, 428, 442, 447, 455, 486, 487, 514, 524, 525, 536, 541, 585, 597, 611, 634, 635, 636, 637, 639, 646, 647, 670, 684, 692, 704, 722, 730, 750, 938
immediate predecessor • 263, 265
impacted • 518, 520
implementation-defined • 9, 22, 23, 29, 35, 40, 52, 55, 56, 57, 59, 60, 61, 64, 69, 75, 79, 82, 85, 87, 88, 89, 90, 91, 93, 99, 104, 110, 120, 121, 122, 123, 127, 128, 129, 130, 135, 137, 138, 140, 150, 152, 153, 156, 174, 175, 187, 188, 190, 191, 201, 262, 263, 264, 319, 321, 323, 328, 339, 348, 349, 356, 380, 442, 443, 446, 447, 448, 450, 454, 455, 492, 507, 518, 519, 536, 548, 551, 559, 561, 564, 567, 569, 570, 571, 573, 574, 575, 576, 613, 616, 626, 629, 634, 640, 645, 652, 656, 661, 665, 667, 669, 670, 681, 684, 692, 700, 701, 702, 704, 705, 706, 711, 712, 737, 742, 744, 749, 761, 762, 765, 768, 769, 772, 774, 777, 780, 781, 783, 784, 785, 786, 788, 828, 829, 853, 875, 877, 886, 887, 888, 901, 903, 910, 912, 923, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953
<implementation-defined character repertoire name> • 22, ***348***, 349, *442*, 443, 949
<implementation-defined collation name> • ***447***, 448, 949
<implementation-defined dictionary name> • ***446***, 448
<implementation-defined routine name> • ***446***, 448, *454*, 455
<implementation-defined translation name> • ***454***, 455, 949
<implementation-defined universal character form-of-use name> • ***348***, 349, 949

implementation-dependent • 9, 13, 14, 15, 21, 42, 43, 46, 47, 48, 49, 65, 67, 68, 82, 83, 84, 87, 89, 90, 91, 111, 112, 146, 162, 174, 201, 209, 219, 225, 239, 242, 250, 251, 252, 274, 275, 323, 338, 339, 342, 364, 370, 384, 394, 409, 421, 427, 434, 437, 450, 523, 525, 529, 536, 537, 543, 551, 570, 577, 578, 585, 586, 591, 604, 612, 613, 619, 623, 624, 650, 652, 656, 661, 674, 696, 697, 700, 701, 707, 711, 712, 713, 741, 751, 752, 753, 757, 784, 787, 788, 874, 955, 956, 957, 958, 959, 960, 961
implicit • 3, 16, 23, 32, 49, 58, 59, 61, 85, 87, 88, 90, 91, 119, 120, 121, 122, 123, 124, 127, 128, 130, 135, 146, 147, 149, 154, 157, 166, 176, 177, 179, 180, 181, 197, 208, 222, 223, 240, 261, 262, 318, 325, 328, 329, 334, 339, 345, 352, 353, 355, 356, 358, 363, 364, 365, 366, 368, 373, 374, 375, 376, 379, 381, 384, 386, 387, 390, 391, 395, 396, 401, 403, 408, 411, 415, 419, 421, 422, 425, 426, 427, 428, 430, 434, 436, 437, 439, 442, 443, 445, 447, 448, 449, 452, 455, 457, 459, 460, 463, 466, 468, 469, 470, 471, 472, 473, 477, 480, 481, 483, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 500, 503, 512, 513, 514, 515, 516, 536, 537, 540, 541, 542, 543, 544, 549, 550, 551, 570, 572, 573, 574, 575, 576, 577, 585, 587, 590, 610, 611, 614, 615, 617, 619, 627, 632, 638, 639, 643, 650, 655, 656, 664, 666, 667, 669, 678, 692, 705, 706, 726, 730, 732, 733, 744, 749, 751, 761, 765, 768, 772, 777, 780, 846, 851, 858, 882, 908, 925, 944, 945, 946, 948, 949, 951, 952, 957, 960
Implicit • 25, 26, 27, 41, 147, 156
IN • 63, *101*, *152*, 245, *280*, *338*, 374, *540*, 550, *604*, 704, *794*, *797*, *798*, *800*, *801*, *804*, *805*, *806*, *807*, *815*, *816*, *817*, *832*, *833*, *836*, *837*, *838*, *839*, *841*, *843*, *845*, *846*, *848*, *849*, *850*, *851*, *852*, *853*, *854*, *856*, *858*, *859*, *860*, *861*, *862*, *863*, *865*, *868*, *870*, *872*, *874*, *876*, *878*, *881*, *883*, *886*, *893*, *894*, *895*, *896*
include • 8, 16, 17, 22, 23, 24, 31, 32, 34, 37, 38, 41, 43, 47, 48, 49, 50, 51, 53, 54, 55, 58, 61, 65, 67, 92, 93, 103, 104, 110, 118, 129, 131, 144, 146, 147, 151, 158, 170, 181, 225, 226, 232, 239, 241, 242, 243, 253, 261, 263, 307, 322, 337, 342, 346, 348, 349, 351, 353, 363, 364, 366, 367, 369, 376, 377, 381, 384, 385, 387, 390, 391, 399, 401, 403, 407, 408, 409, 410, 413, 416, 417, 419, 421, 422, 424, 426, 427, 428, 430, 434, 436, 437, 443, 445, 447, 448, 449, 452, 455, 457, 459, 460, 463, 469, 472, 473, 491, 492, 494, 496, 503, 504, 505, 513, 514, 515, 516, 517, 518, 519, 520, 521, 543, 552, 581, 611, 624, 626, 628, 632, 633, 635, 640, 644, 650, 653, 711, 712, 753, 764, 765, 779, 834, 849, 927, 931, 932, 933, 956, 969, 970
independent • 9, 17, 36, 67, 514, 515, 626, 629, 634, 640, 645, 951
independent node • 514, 515
Index typography • 17
indicator • 63, 64, 65, 99, 134, 136, 137, 138, 139, 140, 239, 317, 318, 322, 338, 339, 340, 548, 696, 697, 700, 904, 905, 923, 945, 947, 949, 958
INDICATOR • *101*, *134*, *139*, *545*, *546*, 689, *695*, 696, 697, 700, 711, 712, 714, 715, *760*, 762, 960
indicator overflow • 318, 904
<indicator parameter> • ***134***, 137, 138, ***139***, 140, 239, 697, 700, 923, 947
<indicator variable> • ***134***, 137, 138, ***139***, 140, 239, 923, 947
Information Schema • 1, 13, 22, 81, 84, 87, 91, 92, 94, 399, 425, 461, 651, 789, 790, 791, 792, 793, 795, 796, 797, 798, 800, 802, 803, 804, 805, 806, 810, 811, 812, 813, 814, 815, 818, 819, 820, 821, 822, 827, 831, 832, 931, 941, 955, 961
INFORMATION_SCHEMA • 22, 59, 91, 92, 122, 124, 349, 548, 789, 790, 791, 792, 793, 794, 796, 797, 798, 800, 804, 805, 806, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 832, 833, 834, 835, 836, 841, 843, 845, 846, 848, 850, 851, 852, 854, 856, 858, 859, 860, 861, 862, 863, 865, 866, 867, 868, 870, 872, 874, 875, 876, 877, 878, 880, 881, 883, 885, 886, 889, 901, 953
inherit • 37, 47, 367, 368, 370, 373, 376, 404, 410, 411, 413, 415, 487, 488, 521, 526, 528, 636, 642, 647, 848
inherited column • 47, 367, 368, 373, 376, 404, 410, 413, 636, 642, 647, 848
<initial expression> • ***260***, 261, 262, 263
INITIALLY • *101*, *352*, 353, 384, 427, 459, *796*, *810*, *814*, 842, *843*, 844, *852*, 853, *862*, 925
INITIALLY_DEFERRED • 796, 810, 814, 842, 843, 844, 852, 853, 862
INNER • 54, *101*, *221*, 222, 223, 227
innermost • 11, 14, 139, 143, 307, 649, 650
INOUT • 63, *101*, 374, *540*, 550
<in predicate> • *271*, **280**, 524, 704, 705, 706
<in predicate value> • ***280***
INPUT • *101*, *709*
INSENSITIVE • 67, 68, *101*, 604, *609*, 610, 611, 613, 615, 616, 626, 629, 634, 640, 645, *721*, *722*, 927, 929, 956
INSERT • 58, 82, 83, *101*, *237*, *338*, *343*, 344, 346, 347, 370, 404, 411, 423, 424, *459*, 461, *465*, 467, 469, 470, 471, 472, 474, 476, 477, 503, 504, 512, 513, 521, *631*, 632, 633, 640, 644, 651, 653, 740, *863*, 864, *868*, 869, *870*, 871, *881*, 882, 925
<insert column list> • 208, 472, ***631***, 632, 633, 635, 636, 705
<insert columns and source> • ***631***, 632, 634, 635, 636, 637, 938
<insert point> • ***631***, 632, 634, *643*, 646

<insert statement> • 67, 71, 74, 75, 76, 78, 144, 147, 207, 208, 210, 231, *237*, 244, 245, 257, 258, 381, 424, 425, 461, *465*, 467, 468, 470, 471, 472, 477, 524, *554*, 625, **631**, 632, 633, 636, 637, *702*, 704, 705, 740, 741, *783*, 921, 924, 928, 933, 937, 938, 960
INSTEAD • *101*, 144, 239, *464*, 473, 474, 475, 478, 966
insufficient item descriptor areas • 710, 712, 908
INT • *101*, *126*, *760*, 761, *774*, 775
INTEGER • 21, 22, 28, 40, *101*, *126*, 127, 128, 182, 261, 262, 264, 275, *528*, 557, 559, 561, 564, 566, 567, 569, 687, 688, 690, 700, 761, 762, 765, 769, *771*, 772, 774, *776*, 777, 780, *829*, *837*, *883*, 946, 950
integrity constraint • 1, 49, 85, 87, 91, 183, 322, 353, 384, 396, 459, 626, 629, 664, 742, 756, 906, 908
integrity constraint violation • 183, 322, 353, 664, 742, 756, 906, 908
<integrity no> • ***18***, 19
<integrity yes> • ***18***, 19
<intermediate> • ***18***, 19
<internal collation source> • 443, *446*, **447**, 449, 450, 877, 918
<internal translation source> • ***454***, 456, 879, 919
INTERSECT • 38, 56, 57, *101*, *247*, 249, 252, 255, 256, 258, 275, 628, 933
INTERVAL • 21, 22, 31, *32*, 33, 66, 90, *100*, *101*, *108*, 111, *126*, 131, 148, 196, 198, 200, 201, 275, 305, 306, *546*, 557, 559, 561, 564, 566, 567, 569, 684, 688, 689, 690, *695*, 697, 700, 701, 704, 705, 711, 713, 714, 715, *794*, *799*, *836*, *837*, *838*, 840
interval data type descriptor • 30, 31
<interval factor> • ***199***, 200
interval field overflow • 181, 182, 319, 321, 905
<interval fractional seconds precision> • 33, 112, 131, 153, ***327***, 328, 329, 688, 705, 712, 713, 948
<interval leading field precision> • 33, 131, 201, 275, ***327***, 328, 329, 688, 705, 712, 713, 948
<interval literal> • *106*, **108**, 111, 112, 113, 114, 380, 931
<interval primary> • ***199***, 200
<interval qualifier> • 31, 33, *108*, 111, 112, *126*, 129, 131, *199*, 201, **327**, 328, 329, 330, 380, 557, 688, 691, 704, 711, 713, 840, 934
<interval string> • *98*, 103, ***108***
<interval term 1> • ***199***, 200, 201
<interval term 2> • ***199***, 200
<interval term> • *196*, 197, ***199***, 200, 705
<interval type> • *125*, **126**, 129, 132, 688, 711, 713, 932
<interval value expression 1> • ***199***, 200, 201
<interval value expression> • *152*, 153, 154, *184*, 185, 186, *196*, 197, 198, ***199***, 200, 201, 202, *684*, 933
INTERVAL_CODE • 840
INTERVAL_PRECISION • 840
intervening • 14, 43, 104, 143, 219, 240, 249, 252, 307, 488
INTO • *101*, *338*, *604*, *605*, *617*, *623*, *631*, *710*, 717, 726
<introducer> • 61, ***106***, 110, *115*, 118
in usage by • 427, 433
invalid catalog name • 677, 907
invalid character set name • 680, 907
invalid character value for cast • 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 905
invalid condition number • 656, 741, 907
invalid connection name • 670, 907
invalid constraint mode state • 657
invalid cursor mode state • 614, 907
invalid cursor name • 703, 719, 722, 724, 907
invalid cursor state • 614, 615, 618, 621, 626, 640, 708, 743, 907
invalid datetime format • 130, 905
invalid descriptor count • 713, 715, 906
invalid descriptor index • 692, 696, 699, 906
invalid enumeration name • 182, 905
invalid enumeration value • 194, 905
invalid escape character • 283, 289, 905
invalid indicator parameter value • 322, 905
invalid limit value • 262, 264, 905
invalid parameter value • 570, 905
invalid return value for predicate • 906
invalid schema name • 678, 907
invalid specification • 661, 663, 667, 907
invalid SQL descriptor name • 524, 692, 694, 696, 699, 710, 907
invalid SQL statement • 618, 621, 626, 633, 640, 644, 754, 907
invalid SQL statement name • 708, 709, 717, 722, 724, 907
invalid target specification value • 640, 907
invalid time zone displacement value • 198, 684, 905
invalid transaction state • 572, 578, 626, 628, 633, 640, 644, 656, 674, 681, 683, 785, 786, 907
invalid transaction termination • 664, 666, 907
invalid update value • 229, 230, 245, 905
<in value list> • ***280***, 281, 704, 706, 925
invisible • 68, 616, 626, 629, 634, 640, 645
IS • 43, *101*, 165, 166, *296*, 297, *309*, *310*, 311, *313*, *373*, 375, 386, *760*, *763*, 764, *767*, 768, *771*, *776*, *779*, *808*, *809*, *822*, *836*, *837*, *838*, *848*, *849*, *876*, *881*, *886*, *900*
<ISO 10646 character name> • ***442***, 443
<ISO 10646 position> • ***442***, 443
ISOLATION • *101*, *655*
isolation level • 86, 87, 88, 91, 92, 572, 655, 656, 785, 951
<isolation level> • ***655***
IS_DEFERRABLE • 796, 810, 814, 842, 843, 844, 852, 853, 862
IS_GRANTABLE • 804, 805, 806, 807, 861, 863, 864, 865, 868, 869, 870, 871, 872, 873, 891, 892, 895, 896
IS_NULLABLE • 799, 848, 849

IS_UPDATABLE • 798, 846, 847
<item number> • ***695***, 696, 697, ***699***, 700
<item reference> • *133*, 134, 135, **139**, *594*
<iteration expression> • ***260***, 261, 263

— J —

JOIN • *101*, *221*, 223, 225, 229, 230, *794*, *796*, *799*, *800*, *801*, *810*, *811*, *812*, *813*, *814*, *818*, *819*, *820*, *821*, *822*, 924
<join column list> • ***221***, 222
join columns • 222, 223, 224, 225, 226, 228, 229
<join condition> • 54, 143, ***221***, 222, 226, 252, 307, 633
<joined table> • 53, 56, *143*, 145, 146, 219, ***221***, 222, 223, 225, 226, 227, 228, 229, 230, 238, *247*, 250, 251, 252, 253, 254, 259, 307, 626, 628, 633, 639, 640, 644, 932, 933
<join specification> • 53, ***221***, 222
<join type> • 53, 54, ***221***, 222, 225, 226

— K —

KEY • 43, 50, 53, *100*, *101*, *221*, 223, 224, 225, 226, 227, 228, 365, 366, 368, 375, 384, *386*, 387, 388, *389*, 390, 417, 626, 628, 633, 639, 644, 689, *695*, 696, 697, 699, 737, *791*, *801*, *813*, *821*, *832*, *833*, *834*, *835*, *838*, *839*, *841*, *843*, *845*, *846*, *848*, *850*, *851*, *852*, *853*, *854*, *856*, *858*, *859*, *860*, *861*, *862*, *863*, *865*, *866*, *867*, *868*, *870*, *872*, *874*, *876*, *878*, *880*, *881*, *883*, *885*, *889*, *899*, *900*, *901*, 931, 935
<key word> • 21, 23, *98*, **99**, 103, 104, 105, 118, 145, 688, 689, 924, 932, 943, 953, 963
KEY_COLUMN_USAGE • 813, 821, 854, 899, 900, 931
KEY_DEGREE_GREATER_THAN_OR_EQUAL_TO_1 • 900
KEY_MEMBER • *100*, 689, *695*, 697, 699
KEY_TYPE • *100*, *695*, 696, 697
known not nullable • 42, 251, 252, 849

— L —

LANGUAGE • *101*, *331*, 540, 541, *801*, *827*, *835*, *861*, *886*, 887, 888, 931, 953
<language clause> • 61, **331**, 332, *535*, 536, *540*, 552, 711, 712, 751, 835, 934, 958
<language name> • ***331***, 552
LAST • *101*, *617*, 618, 619
LEADING • *101*, *155*, 160
LEAVE • *101*, *601*, 966
<leave statement> • 72, *554*, 587, 588, **601**, 602, 921
LEFT • 54, *101*, *221*, 223, 227, 633, 640, 644, *794*, *799*, *800*
<left bracket> • *95*, ***96***, *99*, *287*, 288, 289, 290, 292, 293, *763*, *776*
<left paren> • *18*, *95*, **96**, *125*, *126*, *143*, *149*, *152*, *155*, *162*, *164*, *165*, *168*, *184*, *194*, *199*, *203*, *207*, *211*, *213*, *215*, *221*, *248*, *260*, *268*, *280*, *287*, 288, 289, 290, 292, *296*, *307*, *309*, *327*, *333*, *341*, *343*, *363*, *379*, *386*, *389*, 396, *398*, *421*, 422, *442*, *446*, *447*, *454*, *459*, *465*, *480*, *485*, *486*, *495*, *539*, *553*, *582*, *631*, 637, *638*, *652*, *659*, *747*, 748, *760*, *767*, *774*, *779*, 918
LENGTH • *100*, *102*, *152*, 154, 176, 179, 180, *546*, 557, 558, 687, 688, 689, *695*, 696, 697, 699, 700, 701, 704, 711, 712, 713, 714, 715, 716, *736*, 738, 744, 765, *794*, *799*, *836*, *837*, *838*, 839, 952, 960
<length> • 81, 110, *125*, **126**, 127, 128, 557, 704, 706, *760*, 761, 762, *763*, 764, 765, *767*, 768, *771*, 772, *774*, *776*, 777, 778, *779*, 780, 781, 930
<length expression> • 24, 28, 30, ***152***, 153, 154, 923, 932, 947
LESS • *101*, *485*, *531*, *532*, *533*, *801*, *835*, 966
<less-than clause> • 149, 273, 274, *363*, 371, ***485***, 492, 612, 917
<less-than function specification> • *485*, **486**, 489
<less than operator> • *95*, **96**, *273*, *447*, 449, 612, 788
<less than or equals operator> • *98*, **99**, *273*
LEVEL • *101*, *655*
<level of isolation> • ***655***, 656, 951
<levels clause> • ***421***, 422, 425, 926
LIKE • 25, *101*, *282*, 283, 284, 293, *363*, 364, 650
<like clause> • ***363***, 364, 368, 371, 650, 917
<like predicate> • 25, *271*, **282**, 284, 706
LIMIT • *101*, *260*, 262, 264, 966
<limit clause> • ***260***, 262, 948
<limited collation definition> • 23, ***442***, 443
linear recursive union • 262, 263
LIST • 39, *101*, *126*, 131, *215*, 335, 338, *361*, 506, 533, 632, 634, 643, 919, 966
<list type> • ***126***, 129
<list value constructor> • *109*, 112, 113, **215**, 216
<literal> • 21, 38, 54, 56, **106**, 111, 121, *133*, 137, 172, 173, 174, 175, 177, 179, 180, 181, 182, 183, 238, 341, *379*, 380, 381, 382, *597*, 693, 694, 698, 701, 702, 716, 787, 923, 929, 931
LOCAL • 45, 48, *101*, *117*, 121, 123, *196*, 197, 198, *361*, 369, *421*, 422, 424, *649*, *684*, *845*, *846*, 847, *889*
<local declaration> • ***584***, 585
<local declaration list> • 119, 334, 542, ***584***, 585, 586, 590, 959
<local or schema qualified name> • *115*, ***116***
<local or schema qualifier> • **116**, 119, 333, 334, 363, 541, 542, 649, 650
<local qualified name> • ***116***
<local qualifier> • ***116***, 119, 542
<local table name> • ***115***, 122, 650, 742, 743
local temporary table • 45, 46, 47, 89, 90, 120, 122, 344, 358, 361, 369, 371, 390, 398, 401, 419, 421, 649, 651, 717, 742, 743, 845, 945, 956
LOOP • *101*, *602*, *605*, 966

<loop iteration scheme> • ***602***
<loop statement> • 72, *554*, 585, **602**, 921, 959
<low> • ***18***, 19
LOWER • 24, *101*, *155*, 159, 287, 293, 295
lower case • 24, 97, 105, 159, 293, 295, 764, 789, 931

— M —

<major category> • ***755***, 759, 922
<mantissa> • ***107***, 110, 112, 176, 178
MATCH • 50, 51, *101*, *302*, *389*, 391, 396, *546*, *811*, *856*, 857, 925
matching • 86, 228, 302, 341, 342, 392, 393, 394, 703
matching rows • 86, 228, 302, 392, 393, 394
<match predicate> • *271*, 272, **302**, 303, 304, 391, 706, 925
<match type> • 50, 385, ***389***, 391, 392, 393, 394, 857
<match value> • ***282***, 284, *286*, 287, 288, 289, 706, 934
MATCH_OPTION • 811, 856, 857
MAX • *101*, *149*, 150, 151, 235, 241, 275, *447*, 450, *692*, *794*, *799*, *836*, *837*, *838*, 839, *898*, 951, 958
maximal • 37, 47, 365, 368, 413, 636
maximal supertype • 37, 365
<member> • 365, ***485***
<member list> • ***485***, 486, 488, 491
<member name> • *343*, ***480***
<member renaming element> • ***362***, 366, 368, 413
MESSAGE_LENGTH • *100*, *736*, 738, 744
MESSAGE_OCTET_LENGTH • *100*, *736*, 738, 744
MESSAGE_TEXT • *100*, *736*, 738, 744, 952
MIN • *101*, *149*, 150, 151, 235, 241, 275, *447*, 450, 958
<minus sign> • *95*, **96**, *99*, 103, *107*, *108*, *187*, 188, *196*, *199*, 200, *287*, 288, 289, 290, 292, 329, *447*
MINUTE • 31, *32*, 33, 90, *101*, *102*, 111, 130, 131, 148, *152*, 154, 196, 324, *327*, 684, 691, 701, *838*
<minutes value> • *107*, ***108***, 111
MODIFY • *101*, *447*, 448, 449, *454*, 455, 966
module • 45, 46, 47, 59, 60, 61, 62, 65, 66, 68, 70, 73, 74, 79, 80, 81, 82, 83, 84, 85, 89, 90, 91, 92, 103, 109, 110, 116, 118, 119, 120, 121, 122, 123, 135, 136, 137, 139, 334, 337, 339, 343, 355, 356, 357, 358, 359, 368, 370, 422, 428, 439, 443, 444, 445, 449, 455, 456, 460, 466, 472, 483, 493, 496, 497, 499, 500, 502, 517, 518, 535, 536, 537, 538, 540, 541, 542, 544, 545, 551, 552, 553, 554, 556, 570, 571, 572, 589, 604, 609, 610, 614, 617, 621, 625, 632, 638, 649, 650, 652, 653, 664, 666, 667, 669, 670, 673, 687, 708, 709, 717, 721, 724, 726, 728, 729, 730, 739, 740, 748, 750, 751, 753, 754, 887, 911, 913, 917, 919, 920, 927, 937, 943, 944, 945, 947, 948, 949, 950, 951, 956, 957, 958, 961
MODULE • 46, *101*, *115*, *116*, 119, 334, *343*, 344, 359, 499, *500*, *502*, *535*, 650, 739, 740, 742, 743, *886*, 887
<module> • 45, 46, 47, 59, 60, 61, 62, 65, 66, 73, 74, 79, 80, 81, 82, 83, 84, 85, 89, 90, 91, 110, 118, 119, 120, 121, 122, 123, 135, 136, 137, 139, 334, 337, 339, 356, 368, 370, 422, 428, 439, 443, 444, 449, 455, 456, 460, 466, 472, 483, 493, 496, 499, 500, **535**, 536, 537, 538, 540, 541, 542, 544, 545, 551, 552, 570, 571, 572, 589, 604, 609, 610, 614, 617, 621, 625, 632, 638, 649, 650, 652, 653, 664, 666, 667, 669, 670, 673, 708, 709, 717, 721, 724, 726, 728, 729, 730, 750, 751, 753, 754, 887, 911, 913, 920, 927, 937, 943, 944, 945, 947, 948, 949, 950, 951, 956, 957, 958, 961
<module authorization clause> • 59, 61, 89, 120, ***535***, 538, 751, 943, 945
<module authorization identifier> • 61, 83, 356, 517, ***535***, 536, 537, 570, 652, 653, 669, 945, 948, 951, 961
<module character set specification> • 61, 103, 109, 110, 118, 445, 517, ***535***, 536, 538, 751, 937, 949
<module contents> • 119, 120, 334, ***535***, 536, 538, 937
<module definition> • 61, 70, *355*, 357, 497, **500**, *553*, 556, 740, 917, 919, 920
<module name> • 61, 79, 91, ***116***, 119, 120, 121, 122, 334, *343*, 359, 499, *500*, *502*, *535*, 536, 537, 541, 542, 545, 650, 751, 957, 961
<module name clause> • ***535***, 536, 751
<module remainder> • *500*, ***535***
module variable • 536, 537
monadic • 9, 25, 112, 156, 157, 188
monadic operator • 9, 25, 156, 157
MONTH • 31, 32, 33, *101*, 111, 129, 130, 131, 324, *327*, 328, 691, 705, *838*
<months value> • ***108***, 111, 329
MORE • *100*, 606, 607, *735*, 737, 738
MOVE • *101*, *643*
multiple server transactions • 669, 672, 906
multiset • 9, 10, 21, 38, 39, 42, 44, 45, 109, 112, 113, 126, 129, 131, 170, 173, 174, 211, 213, 214, 219, 225, 226, 227, 242, 253, 275, 310, 370, 422, 505, 508, 628, 904, 907
MULTISET • 39, *101*, *126*, 131, *213*, 335, 338, 908
<multiset type> • ***126***, 129
<multiset value constructor> • *109*, 112, 113, **213**, 214
MUMPS • 3, 5, 66, 79, *100*, *331*, 332, 339, 544, 550, 566, 575, *747*, *748*, 749, 751, 753, 757, *774*, 775, *886*, 887, 888, 912, 934, 950
<MUMPS character variable> • ***774***
<MUMPS host identifier> • *748*, ***774***
<MUMPS length specification> • ***774***, 775
<MUMPS numeric variable> • ***774***, 775
<MUMPS type specification> • ***774***
<MUMPS variable definition> • *748*, **774**, 775

— N —

n-adic operator • 9
NAME • 99, *100*, 102, 535, 540, 545, 547, 680, 688, 689, *695*, 697, 699, 700, 701, 711, 712, 714, 715, 735, 736, 737, 740, 742, 743, 744, 748, 791, 792, 793, 794, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 845, 846, 848, 849, 850, 851, 852, 853, 854, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 885, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 952, 953, 960, 961
<named columns join> • 53, ***221***, 222, 226, 227
NAMES • *102*, *535*, *680*, 740, *748*
NATIONAL • 22, 23, *102*, *125*, 127, 943, 946
<national character string literal> • 103, **106**, 109, 110, 114, 931
<national character string type> • ***125***, 132, 932
NATURAL • 53, *102*, *221*, 222, 223, 225, 226, 227, 410, *545*
natural search order • 264
NCHAR • *102*, *125*, 127
NEW • *102*, *238*, *362*, 364, 368, 370, *464*, *465*, 467, *490*, *594*, 740, 958, 966
<newline> • ***99***, 103, 104, 109, 945
<new object name> • ***594***, 595
<new statement> • 72, 541, *554*, **594**, 595, 740, 921
<new type name clause> • ***362***, 364, 370
<new values correlation name> • *238*, *464*, *465*, **466**, 467, 468, 473, 474, 475
<new values name> • 467, 468
<new values table alias> • ***465***, 467, 468, 474, 475, 476
NEW_TABLE • *102*, *465*, 467, *824*, *881*, 882
NEXT • *102*, *617*, 618, 619, 726
NO • 23, 40, *102*, 274, 349, *389*, 390, 391, 396, *446*, 448, *856*, 857, *876*, 877, 918, 966
No collating sequence • 25, 26, 27, 421
no data • 64, 324, 525, 527, 530, 578, 619, 621, 623, 630, 634, 646, 696, 786, 787, 907, 910
<nondelimiter token> • ***98***, 103
non-deterministic • 49, 235, 241, 252, 398, 422, 460
<nondoublequote character> • ***98***, 103
NONE • 36, *102*, 149, 273, 386, *486*, 488, *531*, *532*, *533*, 612, *638*, 641, 645, 646, *846*, 847, *856*, 857, 966
<non-escaped character> • ***287***, 288, 289
<non-join query expression> • ***247***, *248*, 249, 250, 251, 252, 958
<non-join query primary> • *247*, **248**, 250, 251, 957
<non-join query term> • ***247***, 249, 250, 251, 252, 258, 916, 957, 958
non-linear recursive union • 262, 263
<nonquote character> • *99*, ***106***, 109
<non-reserved word> • ***99***
<non-second datetime field> • ***327***, 329
no outstanding asynchronous SQL statement • 659, 907
no subclass • 903, 904, 905, 906, 907, 908, 909
NOT • 25, 36, 43, *102*, 165, 166, *205*, *278*, *280*, *282*, *286*, 289, 294, *296*, 297, 306, *313*, 339, *352*, 353, *362*, 365, 366, *373*, 375, 384, 385, 386, 387, 388, 427, 428, *447*, 448, 449, 459, 487, *540*, 544, 552, *584*, 585, *755*, 756, 758, *808*, *809*, *822*, *832*, *833*, *834*, *836*, *837*, *838*, *843*, *845*, *846*, *848*, *849*, *852*, *854*, *856*, *861*, *870*, *872*, *874*, *876*, *878*, *881*, *883*, *885*, *886*, *899*, *900*, 925, 935
not available • 906
<not equals operator> • *98*, **99**, *273*
null • 8, 9, 21, 39, 41, 42, 43, 50, 51, 52, 64, 65, 70, 106, 111, 117, 124, 129, 135, 136, 137, 138, 140, 150, 151, 153, 157, 158, 159, 160, 162, 166, 171, 173, 185, 187, 188, 192, 194, 197, 200, 203, 205, 206, 207, 208, 209, 211, 213, 215, 223, 227, 228, 229, 230, 239, 251, 252, 254, 268, 271, 273, 274, 276, 278, 282, 283, 284, 289, 291, 296, 297, 299, 301, 303, 306, 311, 313, 315, 317, 318, 320, 322, 323, 324, 338, 340, 343, 344, 355, 356, 359, 372, 373, 376, 377, 379, 380, 381, 382, 383, 387, 389, 391, 392, 393, 394, 395, 427, 428, 439, 440, 441, 553, 556, 573, 597, 613, 635, 639, 641, 644, 645, 684, 696, 697, 703, 711, 712, 714, 715, 765, 783, 788, 802, 803, 840, 841, 842, 846, 849, 858, 866, 867, 873, 877, 882, 887, 905, 908, 916, 917, 918, 920, 934, 950, 952, 955, 956, 958, 959
NULL • 21, 43, *102*, *165*, 166, *168*, 171, *207*, 228, 229, 239, *296*, 297, 320, 322, *343*, 359, 365, 366, *373*, 375, *379*, 382, 383, 386, 387, 388, *389*, 393, 394, *439*, *441*, 469, 487, *491*, *597*, 626, 628, *798*, *799*, *808*, *809*, *822*, *834*, *836*, *837*, *838*, *843*, *845*, *848*, *849*, *852*, *854*, *856*, 857, *861*, *870*, *872*, *874*, *876*, *878*, *881*, *883*, *885*, *886*, *891*, *892*, *900*, 918, 935
NULLABLE • *100*, 689, *695*, 697, 699, 711, 712, *799*, *848*, 849
<null class definition> • 42, 70, *355*, 356, **439**, 440, *553*, 556, 866, 867, 917, 918, 920
<null class name> • **117**, 124, *343*, 344, 359, *373*, 376, 377, 428, *439*, *441*
<null clause> • *372*, **373**, 376, 377, *427*, 917
NULLIF • *102*, *165*, 703, 705
<null predicate> • *271*, **296**, 297, 703, 916
<null specification> • ***207***, 208, 209, 916
<null state> • *207*, 208, 209, *296*, 297, 320, 322, *379*, 380, 382, 383, *389*, 391, ***439***, 639, 644, 916, 918
<null state list> • ***439***, 440

null value • 8, 9, 21, 39, 41, 42, 50, 65, 106, 129, 137, 140, 150, 153, 158, 159, 160, 166, 171, 173, 185, 188, 192, 194, 197, 200, 206, 208, 209, 227, 228, 229, 230, 274, 283, 284, 291, 296, 297, 299, 303, 306, 311, 317, 318, 320, 322, 338, 340, 381, 382, 393, 394, 395, 440, 613, 635, 641, 645, 696, 697, 712, 715, 783, 788, 840, 846, 858, 887, 905, 908, 916, 950, 952, 955, 956, 959
null value, no indicator parameter • 317, 696, 905
null value eliminated in set function • 150, 908
<null values specification> • ***296***, 297, 916
NULL_STATE • 803, 867
NUMBER • *100*, *735*
<number of conditions> • ***655***, 656, 960
NUMBER_OF_CHARACTERS • 815, 874, 875, 953
NUMERIC • 21, 22, 28, 40, *102*, *126*, 128, *546*, 557, 559, 561, 564, 566, 567, 569, 687, 688, 690, 700, 704, 705, 769, *794*, *799*, 829, *836*, *837*, *838*, 839, 946
numeric data type descriptor • 28
<numeric primary> • ***187***, 188
<numeric type> • ***125***
<numeric value expression> • *156*, *184*, 185, ***187***, 188, *553*
<numeric value function> • **152**, 153, 154, *187*, 923, 932
numeric value out of range • 151, 174, 175, 188, 319, 321, 905
NUMERIC_PRECISION • 794, 799, 829, 836, 837, 838, 839
NUMERIC_PRECISION_RADIX • 794, 799, 829, 836, 837, 838, 839
NUMERIC_SCALE • 794, 799, 836, 837, 838, 839

— O —

<object column> • 395, 396, 470, 471, 472, 610, ***638***, 639, 640, 641, 642, 644, 646, 647, 648, 706, 730, 938
<object column list> • ***638***, 642, 647, 921, 922
object identifier • 9, 18, 35, 40, 41, 111, 129, 131, 141, 164, 170, 172, 173, 174, 185, 319, 321, 324, 374, 381, 396, 488, 489, 544, 557, 558, 559, 561, 564, 566, 567, 569, 570, 594, 595, 635, 911, 915, 957
<object identifier type> • *125*, **126**, 129, 131, 488, 915
<object identifier value expression> • ***184***, 185
<object name> • ***343***, 344, 503, 506, 512, 919, 927, 936
<object parameter name> • ***596***
<object privileges> • ***343***, 344, 503
<object reference> • **146**, 147, 148, *164*, *184*, 185
OBJECT_CATALOG • 794, 802, 803, 806, 815, 816, 817, 824, 826, 872, 873, 881, 882, 885
OBJECT_NAME • 794, 802, 803, 806, 815, 816, 817, 872, 873
OBJECT_SCHEMA • 794, 802, 803, 806, 815, 816, 817, 824, 826, 872, 873, 881, 882, 885
OBJECT_TYPE • 794, 802, 803, 806, 815, 816, 817, 872, 873
<occurrences> • ***692***, 693, 696, 699, 710, 712, 713, 715, 929, 951
octet • 4, 7, 24, 28, 121, 122, 152, 154, 349, 444, 573, 574, 575, 576, 577, 697, 711, 712, 713, 716, 744, 761, 764, 768, 772, 777, 780, 839, 903
<octet length expression> • ***152***, 154
OCTET_LENGTH • *100*, *102*, *152*, 154, 689, *695*, 697, 699, 704, 711, 712, 713, 716, *736*, 738, 744, *794*, *799*, *836*, *837*, *838*, 839, 952, 960
OF • 48, *102*, 144, 239, *362*, 364, 365, 368, 369, 370, *459*, 461, *464*, *465*, 473, 474, 475, 478, *609*, *625*, *638*, *729*, *730*, *732*, *733*, *776*, 777, 778, 930, 958
OFF • 86, *102*, *614*, 615, 633, 640, 644
OID • 9, 10, 35, 36, 38, 41, *102*, *108*, *116*, 129, 141, 164, 170, 238, 239, 250, *362*, 365, 366, 367, 405, 410, 417, 487, 488, 489, 492, *531*, *532*, *533*, 544, 592, 595, 633, 635, 639, 915, 966
<oid literal> • *106*, **108**, 111, 112, 113, 915, 957
<oid options> • ***362***, 365, 368, 370, *485*, 487, 489, 491
<oid string> • ***108***
<oid value> • *108*, **109**, 111, 112, 113, 957
OLD • *102*, *238*, *464*, *465*, 467, 966
<old condition> • ***755***, 756
<old or new values alias list> • *464*, **465**, 467
<old values correlation name> • *238*, *464*, ***465***, 467, 468, 473, 474, 475
<old values name> • 467, 468
<old values table alias> • ***465***, 467, 468, 474, 475, 476
OLD_TABLE • *102*, *465*, 467, *824*, *881*, 882
ON • *100*, *102*, *221*, 225, *237*, *343*, 348, *361*, 366, *389*, 390, 391, 396, 398, 411, 420, 426, 437, 438, 443, 445, 448, 452, 455, 457, *459*, *464*, 466, 469, 481, 497, 499, 502, 504, 505, *614*, 615, *649*, 650, 664, *794*, *796*, *800*, *801*, *810*, *811*, *812*, *813*, *814*, *818*, *819*, *820*, *821*, *822*, *823*, *824*, *825*, *826*, *856*, 857, 918, 965
ONLY • *102*, *372*, 374, 377, 604, *609*, 610, *655*, 656
OPEN • *102*, *604*, *614*, *724*, 739, 740
<open cascade option> • ***614***, 616, *724*, 921
<open statement> • 66, 67, 71, 73, 75, 76, 77, 86, 139, 536, *554*, 578, 610, **614**, 616, 640, 721, 722, 740, 751, 752, 753, 754, 921, 961
<operand data type> • 170, ***486***, 489, 490
OPERATION • *102*, *545*, *548*, *801*, 966
operator • 8, 9, 11, 15, 24, 25, 26, 28, 30, 34, 40, 41, 42, 95, 96, 98, 99, 112, 141, 156, 157, 187, 188, 190, 191, 197, 200, 201, 205, 249, 250, 252, 253, 254, 256, 257, 273, 275, 313, 323, 379, 447, 449, 455, 485, 486, 521, 525, 542, 543, 592, 612, 632, 638, 695, 699, 703, 704, 735, 760, 764, 788, 948
<operator name> • ***485***
<operator name list> • 141, ***485***, 486, 542, 543
OPERATORS • *102*, *485*, 967

OPTION • 48, 82, 83, 91, 92, *102*, 245, 346, 348, *421*, 422, 424, 425, 484, *503*, 504, 505, 507, *508*, 510, 511, *512*, 514, 515, 516, 517, 518, 519, *548*, 636, 641, 646, 743, 789, *798*, *811*, *846*, 847, *856*, 857, 864, 865, 869, 871, 873, 895, 896
OR • 42, 43, 55, *102*, *205*, 206, 293, 294, 306, *794*, *801*, *804*, *805*, *806*, *807*, *836*, *837*, *838*, *839*, *841*, *848*, *850*, *851*, *852*, *856*, *859*, *860*, *863*, *865*, *874*, *876*, *878*, *881*, *886*, *893*, *894*, *895*, *896*, *897*
ORDER • *102*, *609*, 610, 613, 721, 787, 927, 930
<order by clause> • 67, 492, *604*, ***609***, 610, 612, 639, 730, *787*, 788, 956, 959
<ordering specification> • ***609***
originally defined column • 47, 367
OTHERS • *102*, 585, *587*, 967
OUT • 63, *102*, 374, *540*, 550
OUTER • *102*, *221*, *900*
<outer join type> • 54, ***221***
outermost • 14, 143, 585, 590, 604
outer reference • 147, 149, 151, 222, 231, 232, 235, 236, 238, 242, 261, 461, 469, 474, 475, 476, 629, 645, 932
OUTPUT • *102*, 264, 265, 266, *709*
OVERLAPS • 34, *102*, *305*
<overlaps predicate> • 34, 209, *271*, 272, **305**, 306, 703, 705, 934

— P —

<1989 package> • ***18***, 19
PAD • 23, 40, *102*, 274, 349, *446*, 448, *876*, 877
<pad attribute> • ***446***, 448
parameter • 9, 10, 11, 29, 36, 37, 38, 41, 61, 62, 63, 64, 65, 80, 81, 87, 92, 117, 121, 123, 125, 131, 133, 134, 135, 136, 137, 138, 139, 140, 171, 199, 239, 317, 320, 322, 325, 333, 334, 335, 336, 337, 340, 341, 342, 345, 374, 468, 481, 491, 492, 495, 496, 524, 529, 539, 540, 541, 542, 543, 544, 545, 548, 549, 550, 551, 552, 570, 572, 573, 574, 575, 576, 577, 578, 585, 590, 596, 610, 621, 628, 633, 635, 644, 660, 696, 697, 700, 703, 709, 712, 713, 714, 717, 718, 719, 724, 738, 742, 751, 752, 753, 754, 762, 765, 769, 772, 775, 777, 781, 783, 903, 905, 906, 910, 923, 927, 932, 937, 944, 947, 949, 950, 952, 953, 958, 959, 961, 963
<parameter declaration> • 62, 63, 64, 139, 333, 334, 337, 340, ***539***, 541, 542, 543, 544, 545, 548, 549, 551, 572, 578, 585, 590, 610, 751, 752, 753, 754, 927, 937, 961, 963
<parameter list> • 333, 337, ***539***, 541, 542, 543, 544, 548, 549, 552, 596
<parameter mode> • *539*, **540**, 550
<parameter name> • 92, **117**, 123, *133*, *134*, 135, 137, *139*, *539*, 548, 549, 572, 578, 585, 590, *596*, 610, 751, 752, 961
PARAMETERS • *102*, *546*, *801*, 967
<parameter specification> • *133*, ***134***, 136, 137, 322, 628, 633, 644
<parameter using clause> • 713, ***717***, 718
<params> • ***446***, 447, ***454***, 455
<parenthesized style list> • ***539***
PARTIAL • 50, 51, *102*, *302*, 303, *389*, 391, 392, 393, 394, *856*, 857
Pascal • 3, 4, 66, 79, 339, 567, 576, 747, 748, 749, 751, 753, 757, 776, 777, 778, 888, 912, 930, 940, 950
PASCAL • *100*, *331*, 332, 339, 544, 550, 576, 751, 777, 835, *886*, 887, 888
<Pascal host identifier> • *748*, ***776***, 777
<Pascal type specification> • ***776***, 777, 778, 930, 940
<Pascal variable definition> • *748*, **776**, 777
<pattern> • ***282***, 285, *286*, 287, 288, 289, 706, 934
PENDANT • 50, 51, *102*, *389*, 390, 391, 392, 395, 396, 469, 918, 967
pendant reference • 392, 393, 394, 395
<percent> • *95*, **96**, 283, *286*, 288, 289, 290, 292
<period> • *95*, **96**, *107*, *108*, 110, *115*, *116*, *117*, *141*, *146*, 176, 178, *237*, 329, *767*
persist • 10, 45, 46, 47, 49, 58, 59, 61, 68, 69, 85, 90, 358, 361, 369, 390, 398, 650, 651, 653, 845, 893, 894
persistent • 10, 45, 46, 47, 49, 58, 59, 61, 68, 69, 85, 358, 361, 369, 390, 398, 650, 845, 893, 894
persistent table • 893, 894
<PL/I host identifier> • *748*, ***779***, 780
<PL/I type fixed binary> • ***779***
<PL/I type fixed decimal> • ***779***
<PL/I type float binary> • ***779***
<PL/I type specification> • ***779***, 780, 781, 931, 940, 952
<PL/I variable definition> • *748*, **779**, 780, 781, 952
PLI • *100*, *331*, 332, 339, 544, *548*, 550, 577, 751, *824*, 835, *881*, 882, *886*, 887, 888, *890*, *891*, *892*, *897*
<plus sign> • *95*, **96**, *107*, *187*, 188, *196*, *199*, *286*, 288, 289, 290, 292, *442*, 444, *446*, *447*, 918
POSITION • *102*, *152*, 704, *799*, *803*, *813*, *848*, 849, *854*, 855, *867*
<position expression> • 24, 28, 30, ***152***, 153, 154, 704, 923, 947
possible qualifiers • 146
possibly-candidate routine • 334
possibly non-deterministic • 49, 235, 241, 252, 398, 422, 460
possibly nullable • 42, 43, 223, 239, 711, 849
<possibly updatable query expression> • ***247***
precede • 15, 329, 366, 422, 536, 573, 613, 664, 749, 750, 764, 768, 771, 779, 788, 955
PRECISION • 21, 22, 28, 40, *100*, *102*, *126*, 128, *545*, 557, 559, 561, 564, 566, 567, 569, 687, 688, 689, 690, *695*, 697, 700, 701, 711, 712, 713, 714, 715, *760*, 761, 765, *771*, 772, *794*, *799*, 829, *836*, *837*, *838*, 839, 840, 946, 947, 960
<precision> • 81, ***126***, 127, 128, 557, 704, 705, *774*, 775, *779*, 780, 781, 946
predecessor • 128, 263, 265, 439

<predefined type> • 38, ***125***, 131, 250, *362*, 364, 365, 493, 572
<predicate> • *205*, **271**, 272, 284, 492, 523, 925
PREORDER • *102*, *260*, 262, 263, 264, 948, 967
<preparable dynamic delete statement: positioned> • 71, 74, 75, 77, 78, 80, *702*, 703, 718, 719, 720, **732**, 740, 930
<preparable dynamic update statement: positioned> • 71, 74, 75, 77, 78, 80, 424, *702*, 703, 718, 719, 720, **733**, 740, 930
<preparable implementation-defined statement> • 75, ***702***, 952
<preparable SQL data statement> • ***702***, 929
<preparable SQL schema statement> • ***702***, 929
<preparable SQL session statement> • ***702***, 929
<preparable SQL transaction statement> • ***702***, 929
<preparable statement> • 59, 81, 90, 110, 118, 120, 135, 677, 678, 680, **702**, 703, 718, 719, 720, 944, 945
PREPARE • *102*, *546*, *702*, 706, *708*, 719, 739, 740
prepared statement not a cursor specification • 722, 906
<prepare statement> • 16, 59, 73, 78, 80, 81, 90, 110, 120, 123, 135, *555*, 677, 678, 680, **702**, 708, 709, 717, 721, 722, 740, 944, 945
PRESERVE • *102*, *362*, 398, *649*
PRIMARY • 43, 50, 53, *102*, *221*, 223, 225, 226, 227, 365, 366, 368, 384, *386*, 387, 388, 390, 417, *791*, *832*, *833*, *834*, *835*, *838*, *841*, *843*, *845*, *846*, *848*, *850*, *851*, *852*, *853*, *854*, *856*, *858*, *859*, *860*, *861*, *862*, *863*, *865*, *866*, *867*, *868*, *870*, *872*, *874*, *876*, *878*, *880*, *881*, *883*, *885*, *889*, *900*, *901*, 935
primary key • 47, 48, 52, 53, 57, 636, 696, 697, 853, 854, 857, 900
PRIOR • *102*, *617*, 618, 619
PRIVATE • *102*, *372*, *523*, *526*, *529*, 967
privilege • 36, 37, 59, 61, 82, 83, 84, 91, 92, 94, 122, 129, 144, 147, 158, 170, 242, 253, 307, 320, 337, 342, 343, 344, 346, 347, 348, 351, 356, 368, 369, 370, 376, 390, 391, 399, 404, 409, 423, 424, 428, 434, 437, 438, 439, 441, 443, 444, 448, 449, 450, 455, 456, 460, 472, 481, 483, 484, 492, 493, 494, 496, 499, 500, 503, 504, 505, 506, 512, 513, 514, 515, 516, 517, 518, 519, 520, 537, 551, 581, 626, 628, 632, 633, 640, 644, 650, 651, 653, 681, 683, 743, 784, 789, 804, 805, 806, 807, 832, 861, 863, 864, 868, 869, 870, 871, 872, 873, 891, 892, 893, 894, 897, 905, 908, 917, 925, 926, 945, 949, 956
<privilege column list> • *343*, **344**, 346, 347, 504, 512, 917, 925
privilege dependency graph • 514
privilege descriptor • 59, 83, 84, 242, 344, 346, 348, 370, 404, 423, 424, 428, 438, 439, 441, 443, 444, 448, 450, 455, 456, 481, 483, 484, 493, 494, 496, 499, 500, 503, 504, 505, 512, 513, 514, 515, 516, 517, 518, 519, 520, 551, 651, 653, 681, 683, 861, 863, 868, 870, 872, 905
PRIVILEGES • 82, 84, *102*, *343*, 344, 420, 426, 481, 497, 503, 505, 516, 517, 520, 551, *794*, *797*, *800*, *801*, *802*, *803*, *804*, *805*, *806*, *807*, *815*, *816*, *817*, *861*, 863, *868*, *870*, *872*, *891*, *892*, 931
<privileges> • 84, **343**, 344, 376, 390, 391, 423, 428, 449, 455, 494, *503*, 504, 505, *512*, 520, 628
PRIVILEGE_DESCRIPTORS • 546, 863, 891, 892, 893, 894
PRIVILEGE_TYPE • 804, 805, 806, 807, 861, 863, 864, 868, 869, 870, 871, 891, 892, 893, 894
PROCEDURE • *102*, 264, 265, 266, *539*, 540, 751, 752
PROTECTED • *102*, 367, 368, *372*, 967
PUBLIC • 84, 91, *102*, 121, *344*, 346, 348, 367, 368, *372*, 373, 403, 443, 448, 455, 512, 513, 514, 624, 629, 632, 639, 644, 683, 789, *794*, *797*, *800*, *801*, *802*, *803*, *804*, *805*, *806*, *807*, *815*, *816*, *817*, 861, 864, 865, 868, 870, 873, *893*, *894*, *895*, *896*, *897*

— Q —

<qualified identifier> • *115*, ***116***, 119, 121, 143, 333, 352, 460, 470, 483, 493, 495, 500, 543, 549, 652, 742, 743
<qualified join> • ***221***, 222, 230, 916
<qualified local table name> • ***115***, 120, 124, *649*, 650
<qualifier> • ***146***, 149, *237*, 238, 261, 264, 611, 637, 933
<quantified comparison predicate> • *271*, 272, **298**, *307*, 706
<quantified predicate> • 147, *271*, 272, **307**, 308, 525, 916
<quantifier> • ***298***, 299
<query expression> • 14, 17, 45, 48, 49, 52, 54, 55, 56, 60, 91, 144, 147, 207, 210, 241, 242, 245, ***247***, 248, 249, 252, 253, 254, 258, 259, *260*, 261, *268*, 269, 323, 398, 403, 410, 419, *421*, 422, 423, 424, 425, 426, 436, 445, 457, 459, 460, 466, 481, 504, 505, 513, 515, 516, 525, *604*, *609*, 610, 611, 614, 628, 630, *631*, 632, 633, 634, 635, 636, 637, 639, 641, 645, 646, 647, *652*, 653, 704, 705, 724, 730, *787*, 846, 847, 849, 850, 851, 924, 928, 933, 934, 936, 937, 938, 959
query expression too long for information schema • 425, 909
<query primary> • 56, ***247***, 250, 611, 933
<query specification> • 49, 55, 121, 143, 148, **237**, 238, 239, 240, 241, 242, 243, 244, 245, 246, *248*, 249, 252, 269, 309, 398, 421, 422, 425, 460, 524, 525, 611, 623, 625, 637, 639, *702*, 717, 718, 729, 730, 916, 924, 933, 934, 936, 937, 957, 961
<query term> • 56, 121, ***247***, 251, 258, 933
<question mark> • 81, ***96***, *134*
<quote> • *95*, **96**, *99*, 104, *106*, *107*, *108*, 109, 110, 111, 114, *446*, *454*
<quote symbol> • 104, ***106***, 110, 111

— R —

READ • 86, 87, *102*, *372*, 374, 377, 604, *609*, 610, *655*, 656
read-only • 43, 45, 48, 67, 85, 88, 226, 239, 241, 421, 468, 478, 572, 592, 610, 626, 628, 632, 633, 640, 644, 652, 656, 786
READ ONLY • 374, 377, 604, 610, 655, 656
read-write • 85, 88, 572, 656, 785
READ WRITE • 655, 656
REAL • 21, 22, 28, 40, *102*, *126*, 128, *545*, 557, 559, 561, 564, 566, 567, 569, 576, 687, 688, 690, 701, *760*, 761, 765, *771*, 772, *774*, 775, *776*, 777, *836*, 946, 947
recursive • 57, 121, 147, 247, 251, 258, 260, 261, 262, 263, 264, 266, 292, 395, 472, 599, 706, 916, 948
RECURSIVE • *102*, *260*, *890*, *893*, *894*, *895*, *896*, 967
<recursive column list> • ***260***, 261
<recursive union> • 57, 121, 147, *247*, 251, 258, **260**, 261, 262, 263, 264, 266, 706, 916
REF • 21, 35, 36, 40, 41, *102*, *126*, *164*, 324, 366, 374, *486*, 487, 489, *522*, *528*, 967
<reference column list> • ***389***, 390, 396, 397, 410, 469, 918
<reference conversion function> • **164**, *184*, 185, 186
referenced <column name>s • 261
referenced columns • 50, 224, 385, 390, 391, 417, 516, 517
referenced table • 50, 51, 224, 226, 228, 229, 366, 385, 389, 390, 391, 392, 393, 394, 396, 397, 410, 417, 469, 639, 644, 918
<referenced table and columns> • 50, 224, 366, 385, ***389***, 390, 396, 410, 918
REFERENCES • 82, 83, *102*, *343*, 344, 346, 366, 370, *389*, 391, 399, 404, 410, 411, 423, 424, 428, 460, 469, 503, 504, 505, 512, 513, 514, 515, 516, 517, 594, 651, *834*, *835*, *839*, *841*, *843*, *845*, *848*, *850*, *851*, *852*, *854*, *856*, *859*, *860*, *861*, *862*, *863*, 864, *865*, *866*, *867*, *868*, 869, *870*, 871, *872*, *874*, *876*, *878*, *880*, *881*, *883*, *885*, *889*, *901*, 926
<references specification> • 368, *373*, 374, 375, ***389***, 390, 391, 395, 396, 918, 925
REFERENCING • *102*, *237*, *464*, 467, 468, 967
referencing columns • 50, 224, 229, 366, 385, 389, 390, 391, 626, 628, 633, 854
<referencing columns> • 50, 224, 366, 385, ***389***, 390, 626, 628, 633
referencing table • 50, 51, 224, 228, 389, 390, 391, 392, 394, 395, 469
<referencing values clause> • ***237***, 241, *247*, 253, 258
<referential action> • 86, 229, ***389***, 390, 391, 393, 394, 395, 396, 626, 628, 857, 918
<referential constraint definition> • 50, 224, 228, 229, *384*, 385, **389**, 390, 391, 397, 469, 626, 628, 633, 935
<referential triggered action> • 377, ***389***, 397, 925, 935
REFERENTIAL_CONSTRAINTS • 811, 820, 853, 856, 899, 931
<ref function> • ***164***
<regular character set> • *286*, **287**
<regular character set identifier> • ***287***, 288, 289
<regular expression> • ***286***, *287*, 289, 291, 292
<regular expression substring function> • ***155***, 156, 157, 158, 159, 161
<regular factor> • ***286***
<regular identifier> • ***98***, 104, 105, *115*, 789, 922
<regular primary> • ***286***
<regular term> • ***286***
RELATIVE • *102*, *617*, 618, 619
<relative insert point> • ***631***
<release savepoint statement> • 85, *554*, 556, **663**, 740, 920, 922
REPEATABLE • 86, 87, *100*, *655*
repertoire • 8, 10, 22, 23, 25, 40, 61, 91, 92, 93, 103, 109, 110, 118, 127, 130, 137, 152, 156, 157, 159, 169, 172, 173, 175, 176, 177, 178, 179, 190, 275, 323, 348, 349, 379, 442, 443, 448, 449, 452, 520, 765, 874, 904, 949, 957, 961
REPRESENTATION • *102*
request failed • 626, 629, 633, 634, 640, 645, 904
request rejected • 616, 904
requires • 8, 44, 50, 51, 88, 452, 850
reserved • 79, 90, 99, 100, 103, 104, 118, 495, 670, 672, 707, 722, 903, 912, 944, 965
<reserved word> • *99*, **100**, 103, 104, 118, 912, 965
RESIGNAL • *102*, 585, 587, *588*, *607*, 967
<resignal statement> • 65, 73, *555*, 556, 588, **607**, 920, 921
RESTRICT • *102*, *358*, *389*, 393, 394, 396, 410, 417, 419, 426, 436, 481, 497, 498, 518, *546*, *856*, 857, 918
restricted data type attribute violation • 714, 715, 906
restrict violation • 393, 394, 907
<result> • ***165***, 166
<result cast> • 340, ***540***, 549, 551
result constituent • 264, 265, 266
<result data type> • 170, 171, ***486***
<result expression> • ***165***, 166
<result using clause> • 710, 714, 715, ***717***, 718, 929
Retrieval assignment • 317
RETURN • *102*, *260*, 262, 266, 376, *489*, *490*, *491*, *528*, *582*, 967
RETURNED_LENGTH • *100*, 689, *695*, 697, 699, 716
RETURNED_OCTET_LENGTH • *100*, 689, *695*, 697, 699, 716
RETURNED_SQLSTATE • *100*, *604*, *605*, 606, 607, *735*, 737, 742, 743, 744
RETURNS • *102*, 376, *489*, *490*, *491*, *522*, *523*, *526*, *528*, *529*, *531*, *532*, *533*, *540*, 582, *801*, 967
<returns clause> • *539*, **540**, 542, 543, 549, 551, 582
<returns data type> • 335, 336, 337, 338, 340, ***540***, 541, 544, 549, 550, 552

<return statement> • 71, 78, 549, *554*, **582**, 583, 740, 920
<return value> • ***582***
REVOKE • *102*, 411, 420, 426, 437, 438, 441, 445, 452, 457, 481, 497, 499, 502, *510*, 511, *512*, *548*, 740
<revoke role statement> • 70, **510**, 511, *553*, 556, 740, 865, 919, 920
<revoke statement> • 70, 344, 410, 411, 419, 420, 426, 437, 438, 441, 445, 452, 457, 481, 497, 498, 499, 502, **512**, 520, *553*, 740, 861, 863, 936
RIGHT • 54, *102*, *221*, 223, 227, *546*, *548*, 633, 640, 644
<right bracket> • *95*, ***96***, *99*, *287*, 288, 289, 290, 292, 293, *763*, *776*
<right paren> • *18*, *95*, **96**, *125*, *126*, *143*, *149*, *152*, *155*, *162*, *164*, *165*, *168*, *184*, *194*, *199*, *203*, *207*, *211*, *213*, *215*, *221*, *248*, *260*, *268*, *280*, *287*, 288, 289, 290, 292, *296*, *307*, *309*, *327*, *333*, *341*, *343*, *363*, *379*, *386*, *389*, *398*, *421*, 422, *442*, *446*, *447*, *454*, *459*, *465*, *480*, *485*, *486*, *495*, *539*, *553*, *582*, *631*, 637, *638*, *652*, *659*, *747*, 749, *760*, *767*, *774*, *779*
ROLE • *102*, *507*, *511*, *683*, 739, 740, *832*, *833*, *863*, *865*, *889*, *890*, *891*, *892*, *893*, *894*, *895*, *896*, *897*, 967
<role definition> • 70, *355*, 357, **507**, *553*, 556, 740, 833, 865, 917, 919, 920
<role granted> • ***508***, 865
<role name> • **117**, 124, *344*, 346, *507*, *508*, *510*, *511*, *683*, 833, 864, 865, 889
<role revoked> • ***510***
ROLES • 832, 833, 863, 865, 889, 890, 891, 892, 897
ROLE_AUTHORIZATION_DESCRIPTORS • 865, 890, 893, 894, 895, 896
ROLE_COLUMN_GRANTS • 894
ROLE_NAME • 832, 833, 863, 865, 890, 891, 892, 893, 894, 895, 896, 897
ROLE_ROLE_GRANTS • 896
ROLE_TABLE_GRANTS • 893
ROLLBACK • 86, 88, *102*, *547*, *548*, *666*, 740
<rollback statement> • 49, 58, 66, 71, 85, 87, 88, 537, *554*, 571, 577, 584, **666**, 667, 740, 784, 922, 944, 958, 961
ROUTINE • *100*, *102*, *343*, 344, 345, *446*, *454*, *480*, *548*, *736*, 737, 744, *823*, *880*, 967
<routine> • 16, 35, 46, 61, 62, 63, 64, 65, 70, 73, 74, 77, 79, 81, 83, 85, 89, 91, 122, 136, 139, 141, 325, 333, 334, 335, 336, 337, 338, 339, 340, *355*, 356, 369, 482, *486*, 488, 489, 490, 491, 497, *535*, 536, 537, 538, **539**, 540, 541, 542, 543, 544, 545, 548, 549, 551, 552, *553*, 556, 570, 571, 572, 573, 574, 575, 576, 577, 578, 581, 582, *584*, 585, 590, 592, 594, 596, 604, 610, 649, 652, 660, 669, 670, 673, 674, 675, 718, 720, *747*, 751, 752, 753, 754, 785, 913, 917, 920, 944, 948, 949, 950, 951, 958, 961, 963
<routine body> • 61, 62, *539*, **540**
<routine declaration> • *363*, 367, 370, 373, 376, *485*, **486**
routine family • 334, 345, 480, 481, 541, 542, 543
<routine header> • ***539***, 540, 541, 751, 752
<routine invocation> • 62, *133*, 136, **333**, 334, 335, 336, 337, 338, 339, 340, *581*
<routine name> • 61, 62, ***116***, 122, 141, *333*, 334, 335, 339, *343*, 345, *363*, *372*, 374, *480*, 481, *485*, *486*, 489, 490, 491, *539*, 541, 542, 543, 544, 549, 552, 570, 581, 585, 744
ROUTINE_CATALOG • *100*, *736*, 737, 744, *823*, *880*
ROUTINE_NAME • *100*, *736*, 737, 744, *823*, *880*
ROUTINE_SCHEMA • *100*, *736*, 737, 744, *823*, *880*
ROW • *102*, *459*, 461, *465*, 468, 469, 475, 921, 967
ROWS • *102*, *361*, 366, 390, 398, *649*, 650
<row subquery> • *207*, 208, 209, **268**
<row value constructor 1> • ***305***, ***310***
<row value constructor 2> • ***305***, 306, ***310***
<row value constructor> • 43, 53, 186, **207**, 208, 209, *210*, 225, *273*, 274, *278*, *280*, *296*, 297, *298*, 299, *302*, *305*, *310*, 311, 524, 637, *638*, 639, 641, 645, 703, 704, 705, 706, 924, 933, 934, 937, 957
<row value constructor element> • 43, ***207***, 208, 209, 637, 703, 705, 924, 933
<row value constructor list> • ***207***, 209, 210, 637, 957
ROW_COUNT • *100*, *735*, 737, 741, 960

— S —

SAVEPOINT • *102*, *661*, *663*, *666*, 740, 967
<savepoint clause> • ***666***, 667, 922
savepoint exception • 907
<savepoint name> • 85, **117**, 124, 661, 663, *666*, 667
<savepoint specifier> • 85, *661*, *663*, ***666***, 960
<savepoint statement> • 85, *554*, 556, **661**, 740, 920, 922
<scalar subquery> • *184*, 185, 186, **268**, 933
SCALE • *100*, 687, 688, 689, *695*, 697, 700, 701, 711, 712, 713, 714, 715, *794*, *799*, *836*, *837*, *838*, 839, 952, 960
<scale> • 81, ***126***, 127, 128, 557, *774*, 775, *779*, 780, 946
SCHEMA • 22, 59, 89, 91, 92, *99*, *100*, *102*, 120, 122, 124, *343*, 344, 349, *355*, *358*, 503, *535*, 536, 538, *547*, *548*, *678*, 688, 689, *695*, 697, 699, 700, 701, 711, 712, 714, 715, *735*, *736*, 737, 739, 740, 742, 743, 744, 751, 789, *790*, *791*, *792*, *793*, *794*, *796*, *797*, *798*, *799*, *800*, *801*, *802*, *803*, *804*, *805*, *806*, *807*, *808*, *809*, *810*, *811*, *812*, *813*, *814*, *815*, *816*, *817*, *818*, *819*, *820*, *821*, *822*, *823*, *824*, *825*, *826*, *827*, 830, *831*, *832*, *833*, *834*, *835*, *836*, *837*, *838*, *839*, 840, *841*, 842, *843*, *845*, *846*, *848*, *849*, *850*, *851*, *852*, *853*, *854*, *856*, 857, *858*, *859*, *860*, *861*, *862*, *863*, 864, *865*, *866*, *867*, *868*, 869, *870*, 871, *872*, 873, *874*, 875, *876*, 877, *878*, 879, *880*, *881*, 882, *883*, *885*, *886*, *889*, *890*, *891*, *892*, *893*, *894*, *895*, *896*, *897*, *898*, *899*, *900*, *901*, 931, 952, 953

<schema authorization identifier> • 83, ***355***, 356
<schema character set name> • ***442***, 443
<schema character set specification> • 110, 118, ***355***, 356, 357, 375, 519, 935, 949
<schema collation name> • ***446***, 448
<schema definition> • 59, 69, 83, 110, 118, 119, 120, 334, 352, **355**, 356, 363, 370, 375, 421, 427, 439, 442, 443, 447, 455, 459, 466, 483, 495, 500, 519, 541, 542, *553*, 740, 830, 832, 949
<schema element> • 344, ***355***, 356, 503, 917, 925, 935
<schema name> • 17, 46, 58, 59, 61, 81, 89, 90, **115**, *116*, *117*, 119, 120, 121, 122, 123, 124, 334, 349, 352, *355*, 356, 357, *358*, 363, 368, 375, 402, 408, 419, 421, 422, 426, 427, 428, 430, 434, 436, 439, 441, 442, 443, 445, 447, 448, 449, 452, 455, 457, 459, 460, 463, 466, 472, 478, 481, 483, 493, 494, 495, 496, 497, 498, 500, 502, *535*, 536, 541, 542, 543, 549, 551, 650, 651, 652, 677, 678, 742, 743, 744, 751, 898, 935, 944, 945, 949, 956
<schema name clause> • 120, ***355***, 357, 935, 945
<schema qualified name> • 46, 59, 90, ***115***, *116*, *117*, 120, 121, 542, 678, 945
SCHEMATA • 793, 796, 798, 799, 810, 811, 812, 813, 814, 818, 819, 820, 821, 822, 823, 824, 825, 826, 834, 835, 839, 841, 843, 845, 848, 850, 851, 852, 856, 859, 860, 862, 866, 874, 876, 878, 880, 881, 931
<schema translation name> • ***454***, 455
SCHEMA_NAME • *100*, *547*, *735*, 737, 742, 743, *793*, *796*, *798*, *799*, *810*, *811*, *812*, *813*, *814*, *818*, *819*, *820*, *821*, *822*, *823*, *824*, *825*, *826*, *834*
SCHEMA_OWNER • 793, 796, 798, 799, 810, 811, 812, 813, 814, 818, 819, 820, 821, 822, 823, 824, 825, 826, 834
scope • 1, 13, 16, 46, 80, 81, 88, 117, 121, 122, 123, 124, 143, 146, 147, 149, 240, 241, 245, 253, 261, 307, 338, 361, 366, 401, 460, 467, 468, 491, 548, 606, 607, 628, 639, 644, 649, 692, 694, 706, 708, 709, 710, 717, 722, 724, 730, 750, 761, 957
scope clause • 143, 147, 245, 307, 692, 694
<scope option> • ***117***, 121, 123, 692, 694, 706, 710
SCROLL • *102*, *609*, 610, 613, 617, *721*, *722*, 726, 927, 930, 937
SEARCH • *102*, *260*, *548*, 967
<search clause> • ***260***, 261
<search condition> • 17, 43, 50, 51, 54, 55, 62, 86, 144, 147, *165*, 166, *221*, 222, 226, *231*, 232, 234, *235*, 236, *307*, 308, **315**, 376, 385, 386, *398*, 399, 403, 410, 419, 426, 427, 428, 433, 436, 445, 457, *459*, 460, 461, *465*, 468, 472, 474, 475, 476, 481, 504, 514, 516, 517, *599*, 600, *602*, *628*, 629, 630, 634, *643*, 644, 645, 646, 647, 741, 858, 859, 860, 884, 926, 928, 933
search condition too long for information schema • 399, 461, 909
<searched case> • ***165***, 166
<searched when clause> • ***165***, 166
<search order> • ***260***
SECOND • 31, 32, 33, *102*, 111, 130, 131, 153, 324, *327*, 328, 329, 691, 701, 705, *838*, 947
<seconds fraction> • ***108***, 111, 112, 113, 329, 923
<seconds integer value> • ***108***, 111, 329
<seconds value> • ***108***, 111
SECTION • *102*, *748*
SELECT • 54, 55, 56, 82, 83, 91, 92, *102*, 144, 147, 221, 223, 225, 227, *237*, 242, 249, 251, 253, 307, *343*, 344, 346, 347, 368, 369, 370, 385, 386, 387, 391, 404, 410, 423, 428, 472, 503, 504, 512, 513, 514, 516, 517, 521, *623*, 650, 651, 653, 739, 740, 741, 789, *791*, *792*, *793*, *794*, *796*, *797*, *798*, *799*, *800*, *801*, *802*, *803*, *804*, *805*, *806*, *807*, *808*, *809*, *810*, *811*, *812*, *813*, *814*, *815*, *816*, *817*, *818*, *819*, *820*, *821*, *822*, *823*, *824*, *825*, *826*, *827*, *832*, *833*, *839*, *841*, *845*, *846*, *848*, *849*, *850*, *851*, *852*, *853*, *854*, *856*, *858*, *859*, *860*, *863*, 864, *865*, *868*, 869, *870*, 871, *872*, *874*, *876*, *878*, *890*, *891*, *892*, *893*, *894*, *895*, *896*, *897*, *898*, *899*, *900*, 904, 917
<select list> • 55, 56, 92, 143, 144, 148, 149, 222, 225, 231, ***237***, 238, 239, 241, 242, 243, 246, 251, 611, *623*, 624, 696, 697, 703, 709, 710, 715, 718, 924
<select statement: single row> • 71, 74, 75, 77, 121, 143, 144, 147, 148, 309, *554*, 578, **623**, 624, 740, 959
<select sublist> • ***237***, 238, 246, 933
<select target list> • ***623***, 624, 959
<semicolon> • *95*, **96**, *465*, *485*, *486*, *539*, *584*, *589*, *590*, *747*, 749, *763*, *774*, *776*, *779*, *783*
SENSITIVE • 68, *102*, *609*, 613, 616, 626, 629, 633, 640, 645, 921, 956, 967
<sep> • ***442***
<separator> • 98, ***99***, 103, 104, *106*, *107*, 109, 110, 111, 118, 748, 756, 771
sequence • 8, 10, 13, 15, 23, 24, 25, 26, 27, 33, 40, 44, 61, 62, 85, 87, 89, 104, 109, 112, 114, 123, 136, 147, 150, 156, 157, 169, 185, 191, 233, 238, 260, 261, 263, 265, 266, 273, 274, 275, 282, 283, 284, 288, 291, 292, 293, 298, 302, 323, 349, 351, 359, 376, 421, 446, 447, 448, 449, 450, 452, 483, 492, 495, 582, 611, 632, 756, 761, 764, 768, 780, 877, 905, 918, 949
SEQUENCE • *102*, 262, 263, 264, 266, *447*, 448, 449, *546*, *876*, 877, 967
<sequence column> • ***260***, 261, 263, 266
SERIALIZABLE • 86, 87, *100*, 572, *655*, 785
serialization failure • 87, 908
SERVER_NAME • *100*, *735*, 737, 744
SESSION • *102*, *133*, 135, 137, 138, *379*, 380, 381, 382, 383, 398, 459, 615, *681*, 741, 787, *889*, *890*, 932, 935, 947, 967
SESSION_USER • *102*, *133*, 135, 137, 138, *379*, 380, 381, 382, 383, 398, 459, 615, 787, 932, 935, 947

SET • 38, *102*, *125*, *126*, 127, 131, 132, 135, *211*, 228, 229, *260*, 335, 338, *343*, 348, *355*, 359, *361*, 364, *372*, 380, *389*, 391, 393, 394, *406*, *431*, *442*, 443, *445*, 469, *490*, 506, 532, 571, *592*, 615, 622, 626, 628, *638*, *643*, *655*, *657*, 664, *672*, *677*, *678*, *680*, *681*, *683*, *684*, *699*, *730*, *733*, *760*, 761, *763*, 764, *767*, 768, *771*, 772, *776*, 777, *779*, 780, 785, *828*, *856*, 857, *872*, 908, 919, 932, 936, 946
<set catalog statement> • 59, 72, 81, 90, *554*, **677**, 706, 740, 927, 928, 929, 931
<set clause> • 610, ***638***, 639, 640, 641, 643, 644, 646, 647, 706, 928
<set clause list> • 461, ***638***, 639, *643*, 644, 647, *730*, *733*
<set column default clause> • *405*, **406**, 936
<set connection statement> • 72, 88, 89, 91, *554*, 571, **672**, 673, 740, 744, 785, 928
<set constraints mode statement> • 49, 71, 353, *554*, 615, 622, **657**, 658, 740, 927, 928, 929, 931
<set count> • ***699***
<set descriptor information> • ***699***
<set descriptor statement> • 73, 78, 80, *555*, **699**, 700, 701, 740, 960
<set domain default clause> • *430*, **431**, 926
set function • 8, 14, 43, 55, 141, 142, 149, 150, 151, 185, 222, 231, 232, 235, 236, 238, 239, 241, 242, 243, 246, 374, 377, 398, 460, 521, 523, 525, 527, 592, 611, 639, 642, 643, 644, 646, 647, 703, 908, 923, 924, 932
<set function> • ***372***, 374
<set function specification> • 14, 55, **149**, 150, 151, *184*, 185, 222, 231, 232, 235, 236, 238, 239, 241, 242, 243, 246, 398, 460, 523, 525, 611, 639, 643, 703, 924, 932
<set function type> • ***149***
<set item information> • ***699***, 700
<set local time zone statement> • 72, 90, *554*, **684**, 685, 706, 740, 938, 940
<set names statement> • 72, 81, 90, *554*, **680**, 706, 740, 927, 929, 931
set-processing mode • 86, 392, 395, 396, 618, 621, 626, 640
<set quantifier> • ***149***, *237*, 241, 242, 243, 244, 246, 611, *623*
<set role statement> • 84, 346, *554*, 556, **683**, 740, 889, 920, 922
<set schema statement> • 59, 72, 81, 90, *554*, **678**, 706, 740, 927, 928, 929, 931
<set session authorization identifier statement> • 72, 83, *554*, **681**, 682, 706, 741, 937, 938, 940
<set time zone value> • ***684***
<set transaction statement> • 65, 71, 85, 86, *554*, 572, **655**, 656, 741, 785, 937, 938
<set type> • ***126***, 129, 150
<set value constructor> • *109*, 112, 113, **211**, 212
<sign> • ***107***, *108*, 112, *187*, *199*, 200
SIGNAL • *102*, *606*, 967
<signal statement> • 65, 73, *555*, 556, **606**, 920, 921
<signed integer> • ***107***
<signed numeric literal> • *106*, **107**, 112, 174, 175, 380
significant • 28, 29, 31, 33, 34, 40, 67, 174, 175, 176, 179, 181, 182, 188, 197, 200, 201, 223, 275, 324, 327, 328, 329, 380, 573, 616, 626, 629, 634, 640, 645, 771, 777, 943, 951
SIMILAR • *102*, 158, 159, *286*, 289, 291, 293, 294, 295, 967
<similar pattern> • ***286***, 288, 291, 292
<similar predicate> • *271*, 272, **286**, 288, 293, 295, 916
<simple case> • ***165***, 166
<simple comment> • ***99***, 104, 704, 719
<simple comment introducer> • ***99***
<simple Latin letter> • 24, ***95***, *115*, 292, 293, 295
<simple Latin lower case letter> • ***95***, 97, 105, 159, 293, 295, 931
<simple Latin upper case letter> • ***95***, 97, 105, 159, 295, *755*, 903, 953
<simple table> • 56, ***248***, 249, 250, 252, 253, 254, 258, 611, 924, 957
<simple target specification 1> • ***695***, 696, 960
<simple target specification 2> • ***695***, 696, 960
<simple target specification 3> • ***695***, 696, 960
<simple target specification> • **134**, 135, 138, 549, 592, 640, 661, 663, *666*, 667, 687, 688, *695*, 696, 706, *735*, 736, 738, 741, 960
<simple value specification 1> • ***699***
<simple value specification 2> • ***699***, 700
<simple value specification> • 80, 88, *117*, *118*, 121, **133**, 135, 137, 198, 549, *617*, 618, *655*, 670, 687, 688, *692*, 694, *695*, *699*, *702*, 706, 710, 722, *736*
<simple when clause> • ***165***
simply contain • 14, 25, 43, 119, 144, 147, 149, 150, 153, 186, 188, 192, 194, 196, 197, 198, 199, 200, 207, 208, 209, 238, 239, 240, 249, 252, 273, 278, 380, 455, 487, 523, 524, 541, 542, 582, 585, 590, 610, 612, 614, 703, 704, 705, 724, 847, 924, 933
<single datetime field> • ***327***, 328, 329
SIZE • *102*, *655*
SMALLINT • 21, 22, 28, 40, *102*, *126*, 128, *545*, 548, 557, 559, 561, 564, 566, 567, 569, 687, 688, 690, 700, *760*, 761, 765, 769, 780, *837*, 946, 950
<solidus> • *95*, **96**, *99*, 103, 104, *187*, 188, *199*
SOME • *102*, *298*, *307*, *638*, 641, 645, 646
<some> • ***298***, 299
<sort key> • ***609***, 610, 612
<sort specification> • 261, ***609***, 610, 611, 612, 613, 787, 788, 963
<sort specification list> • 147, *260*, 263, 264, ***609***
<source character set specification> • ***454***, 455, 514
SOURCE_CHARACTER_SET_CATALOG • 817, 878, 879
SOURCE_CHARACTER_SET_NAME • 817, 878, 879

SOURCE_CHARACTER_SET_SCHEMA • 817, 878, 879
SPACE • 23, 40, *102*, 349, *446*, 448, *876*, 877, 967
<space> • 22, 40, 80, 88, *95*, **96**, *99*, 103, *107*, *108*, 160, 172, 173, 174, 175, 176, 177, 179, 192, 274, 284, 318, 320, 321, 382, 383, 443, 573, 574, 575, 576, 577, 711, 742, 743, 744, 749, 771
spans • 89, 424
SPECIFIC • *100*, *102*, *343*, *480*, *539*, *546*, *547*, *736*, 737, 744
<specific name> • **116**, *343*, 345, *480*, *539*, 543, 544, 744
SPECIFIC_NAME • *100*, *736*, 737, 744
specified search order • 262, 263, 264, 265
specifies • 1, 2, 11, 13, 18, 22, 24, 29, 31, 32, 33, 34, 35, 37, 38, 41, 43, 45, 49, 52, 53, 54, 55, 56, 58, 62, 63, 64, 65, 66, 67, 80, 82, 83, 84, 85, 86, 89, 104, 111, 118, 127, 128, 129, 131, 136, 137, 138, 139, 149, 150, 151, 164, 166, 171, 199, 200, 206, 223, 226, 228, 229, 235, 239, 241, 249, 253, 273, 296, 324, 328, 329, 331, 336, 339, 344, 346, 347, 349, 365, 367, 368, 370, 373, 375, 376, 377, 379, 381, 382, 383, 384, 386, 387, 390, 391, 393, 394, 396, 398, 399, 404, 427, 428, 438, 441, 448, 449, 456, 460, 461, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 481, 487, 488, 489, 496, 503, 504, 512, 515, 516, 536, 537, 538, 540, 541, 542, 543, 544, 545, 550, 551, 570, 572, 573, 574, 575, 576, 577, 582, 586, 587, 590, 594, 596, 610, 611, 612, 615, 616, 618, 619, 623, 626, 628, 632, 633, 639, 640, 641, 642, 644, 646, 647, 653, 664, 669, 690, 691, 705, 706, 708, 711, 712, 721, 738, 749, 751, 752, 753, 754, 756, 757, 758, 761, 762, 764, 765, 769, 772, 775, 777, 781, 784, 788, 828, 884, 903, 910, 911, 912, 917, 918, 923, 925, 929, 932, 934, 937, 947, 949, 956, 957, 958, 969, 970
SQL • *102*, *747*
SQL-agent • 65, 79, 84, 86, 88, 89, 91, 113, 537, 538, 570, 571, 572, 578, 655, 656, 657, 664, 666, 670, 672, 673, 674, 675, 784, 785, 937, 944, 949, 950, 956, 958, 960
SQL-client • 88, 89, 91, 670, 904, 957
SQLCODE • 64, 65, *102*, *539*, 544, *545*, 557, 559, 561, 564, 566, 567, 569, 738, 741, 750, 751, 752, 753, 754, *760*, 762, 765, 769, 772, 775, 777, 781, 910, 944, 949, 950, 952, 953, 956, 963
<SQL condition> • ***755***, 756, 759, 922
<SQL conformance> • ***18***
SQL-connection • 69, 72, 73, 75, 76, 78, 88, 89, 123, 669, 670, 672, 673, 674, 675, 904, 951, 960
<SQL connection statement> • 91, 466, *553*, **554**, 571, 584, 744, *783*, 784, 785, 927, 931
<SQL control statement> • *553*, **554**, 556, 920
<SQL data change statement> • 473, ***554***, 578
<SQL data statement> • 467, *553*, **554**, 578
<sql diagnostics information> • ***735***
<SQL diagnostics statement> • 65, 91, *553*, **555**, 556, 571, 578, 920, 937
<SQL dynamic data statement> • 85, ***555***, 578, 944
<SQL dynamic statement> • 123, 466, *553*, **555**, 710, 927, 937
<SQL edition> • ***18***
<SQL embedded language character> • ***95***
SQL-environment • 59, 60, 88, 89, 91, 536, 537, 670, 944, 950, 951
SQLERROR • *102*, *755*, 758, 963
SQLEXCEPTION • *102*, *755*, 756, 758, 759, 922, 967
<SQL executable statement> • ***553***, 556, 920
<SQL language character> • 22, 61, ***95***, 103, 109, 118, 172, 173, 175, 176, 177, 178, 179, 536, 749, 946, 949
<SQL language identifier> • **115**, *117*, 118
<SQL language identifier part> • ***115***, 118
<SQL language identifier start> • ***115***, 118
<SQL object identifier> • **18**
<SQL parameter> • ***539***, 548, 549
<SQL prefix> • ***747***, *748*, 749
<SQL procedure statement> • 16, 62, 64, 65, 73, 74, 77, 79, 81, 91, 147, 353, 369, 392, *465*, 523, 529, *540*, 549, **553**, 555, 556, 571, 578, *584*, 586, 588, 601, 618, 626, 640, 659, 660, 666, *747*, 752, 753, 756, 757, 784, 920, 927, 937, 945, 950, 961
<SQL provenance> • ***18***
<SQL routine body> • ***540***, 541, 549
<SQL schema definition statement> • ***553***, 556, 920
<SQL schema manipulation statement> • ***553***
<SQL schema statement> • 59, 466, 503, ***553***, 572, 653, *702*, *783*, 786, 789, 927, 931, 937, 940
SQL-server • 88, 89, 91, 117, 122, 669, 670, 672, 673, 744, 904, 944, 951, 957
<SQL-server name> • 88, **117**, 122, *669*, 670, 744, 944, 951
SQL-session • 10, 32, 45, 46, 47, 59, 60, 61, 68, 69, 72, 73, 74, 76, 78, 80, 81, 82, 83, 84, 87, 88, 89, 90, 91, 110, 111, 113, 118, 120, 123, 130, 135, 137, 147, 162, 181, 198, 356, 363, 436, 438, 472, 478, 481, 496, 497, 498, 500, 502, 507, 510, 511, 537, 551, 570, 571, 572, 649, 650, 651, 652, 653, 657, 669, 670, 672, 673, 674, 677, 678, 680, 681, 683, 684, 700, 744, 784, 785, 865, 889, 890, 891, 892, 943, 944, 945, 949, 950, 951, 956, 957
<SQL session statement> • 466, *553*, **554**, 556, *702*, *783*, 920, 927, 931, 937, 940
<SQL special character> • ***95***, *98*
SQLSTATE • 64, 65, 92, *100*, *102*, 339, 340, 353, *539*, 544, *545*, *546*, *547*, *548*, 557, 559, 561, 564, 566, 567, 569, *589*, *604*, *605*, 606, 607, 660, *735*, 737, 738, 741, 742, 743, 744, 750, 751, 752, 753, 754, *755*, 756, 757, 758, 759, *760*, 762, 765, 769, 772, 775, 777, 781, 903, 904, 906, 910, 922, 953, 956
<SQLSTATE char> • ***755***, 756
<SQLSTATE class value> • ***755***, 756, 757, 758

SQL-statement • 1, 2, 13, 14, 15, 16, 49, 58, 59, 65, 67, 68, 69, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 137, 162, 209, 239, 250, 251, 252, 353, 356, 392, 394, 395, 396, 461, 464, 473, 475, 476, 537, 553, 572, 586, 659, 669, 672, 708, 738, 741, 742, 749, 750, 755, 784, 789, 943, 947, 950, 955, 956, 957, 960, 961
<SQL statement list> • ***584***, 586, *587*, 588, *597*, 598, *599*, 600, *602*, *604*
<SQL statement name> • 16, 77, **116**, 124, *702*, 706, 707, *708*, *709*, *717*, 931
SQL statement not yet complete • 660, 907
<SQL statement variable> • 16, 77, ***702***, 703, *719*
<SQLSTATE subclass value> • ***755***, 756, 757, 758
<SQL terminal character> • **95**
<SQL terminator> • ***747***, *748*, 749
SQL-transaction • 10, 49, 58, 59, 66, 68, 69, 71, 73, 74, 76, 78, 85, 86, 87, 88, 89, 91, 92, 124, 392, 395, 396, 425, 461, 477, 571, 572, 578, 614, 615, 618, 621, 622, 626, 627, 628, 630, 633, 636, 640, 642, 644, 647, 655, 656, 657, 661, 663, 664, 665, 666, 667, 669, 672, 681, 683, 707, 754, 785, 786, 943, 944, 951, 956, 960
<SQL transaction statement> • 466, *553*, **554**, 556, *702*, *783*, 920, 927, 931, 937
<SQL variable declaration> • 139, 379, *535*, 536, 538, *584*, 585, 586, **590**, 591, *747*, 750, 752, 753, 754, 920, 959
<SQL variable name> • **117**, 123, *133*, *134*, 135, *139*, 536, 585, *590*, 592, 750
<SQL variable name list> • ***590***
<SQL variant> • ***18***
SQLWARNING • *102*, *755*, 756, 758, 759, 922, 967
SQL_IDENTIFIER • 791, 828, 832, 833, 834, 835, 836, 841, 843, 845, 846, 848, 850, 851, 852, 854, 856, 858, 859, 860, 861, 862, 863, 865, 866, 867, 868, 870, 872, 874, 876, 878, 880, 881, 883, 885, 889, 901
SQL_LANGUAGES • 827, 886, 931
SQL_LANGUAGE_CONFORMANCE • 827, 886, 887
SQL_LANGUAGE_IMPLEMENTATION • 827, 886, 887, 953
SQL_LANGUAGE_INTEGRITY • 827, 886, 887
SQL_LANGUAGE_SOURCE • 827, 886, 887, 953
SQL_LANGUAGE_YEAR • 827, 886, 887
SQL_TEXT • 22, 31, 35, 104, 118, 127, 135, 348, 380, 550, 689, 828, 875, 877, 946, 953
<standard character repertoire name> • ***348***, 349, *442*, 443, 949
<standard collation name> • *446*, **447**, 448, 450, 949
<standard translation name> • ***454***, 455, 949
<standard universal character form-of-use name> • ***348***, 349, 949
<start field> • 201, 324, ***327***, 328, 329
<start position> • *155*, **156**, 157, 158, 160
statement completion unknown • 89, 908
<statement information> • ***735***
<statement information item> • ***735***, 736, 738
<statement information item name> • ***735***, 736, 737
<statement label> • ***584***, 585, 587, 588, *601*
<statement name> • 80, *116*, **117**, 123, 707, 708, 709, 717, *721*, 724, 960
<statement or declaration> • ***747***, 754
<status parameter> • 333, ***539***, 541, 542, 543, 544, 549, 621
Store assignment • 320
<stored column> • 366, ***372***, 374, 486
string data, right truncation • 172, 173, 176, 177, 178, 179, 180, 192, 193, 318, 320, 321, 905, 909
<string length> • *155*, **156**, 157, 158, 160
<string value expression> • *152*, 153, 154, 158, *184*, 185, **190**, 192, 789
<string value function> • **155**, 161, *190*, 191
STRUCTURE • *102*, 967
SUBCLASS_ORIGIN • *100*, *735*, 737, 742, 952
subject schema name • 541
subject table • 58, 403, 411, 461, 466, 469, 470, 471, 472, 473, 517
<subquery> • 14, 77, 121, 144, 149, 185, 231, 232, 235, 236, 238, 239, 246, ***268***, 269, 309, 398, 399, 421, 460, 472, 611, 628, 629, 632, 640, 644, 645, 924, 926, 928, 930, 933, 934
SUBSTRING • 24, *102*, *155*, 156, 177, 179, *546*, 573, 574, 575, 576, 704
substring error • 158, 160, 905
subtable • 46, 47, 48, 362, 363, 365, 366, 367, 368, 369, 370, 371, 373, 376, 384, 387, 404, 410, 411, 413, 415, 420, 437, 492, 506, 627, 630, 636, 642, 647, 901, 917
<subtable clause> • 46, ***362***, 365, 366, 367, 368, 369, 370, 373, 376, 384, 387, 492
subtable family • 47, 368, 627, 630
<subtable member name> • *362*, **363**, 368, 413
<subtable of existing type> • ***362***
<subtable of new type> • ***362***, 365, 366
subtable row family • 47, 627, 630, 642, 647
subtype • 9, 36, 37, 39, 40, 131, 333, 335, 336, 485, 486, 487, 545, 943
<subtype clause> • 36, ***485***, 486, 487
<subtype component name> • ***485***, 487
subtype family • 37, 333, 335, 336
SUB_TABLES • 901
SUB_TABLE_NAME • 901
successful completion • 64, 578, 757, 903, 907, 910
SUM • *102*, *149*, 150, 151, 947
supertable • 47, 48, 362, 363, 367, 368, 369, 370, 373, 376, 384, 401, 402, 404, 410, 411, 413, 415, 437, 506, 636, 642, 647, 901, 918
<supertable clause> • ***362***, 368, 369, *413*, *415*
<supertable member name> • *362*, **363**, 368, 413
<supertable name> • *362*, **363**, 367, 368, 370, 373, 413, 415
supertype • 36, 37, 324, 325, 335, 336, 365, 368, 484, 485, 487, 488, 522, 528
<supertype clause> • ***485***, 487
<supertype component name> • ***485***, 487
SUPER_TABLE_NAME • 901
synchronous • 79, 571, 577, 578, 659, 660, 666, 904, 906, 907, 950

syntax error or access rule violation • 14, 15, 524, 703, 719, 743, 784, 908
syntax error or access rule violation in direct SQL statement • 15, 743, 784, 908
syntax error or access rule violation in dynamic SQL statement • 15, 703, 719, 743, 908
_SYSTEM • 84, 348, 370, 411, 420, 423, 424, 426, 428, 437, 438, 439, 441, 444, 445, 450, 452, 456, 457, 481, 483, 484, 494, 496, 497, 500, 502, 504, 505, 513, 514, 515, 551, 651, 653
<system descriptor statement> • ***555***
SYSTEM_USER • *102*, *133*, 135, 137, 138, 239, *379*, 380, 381, 382, 383, 398, 459, 615, 787, 932, 935, 947

— T —

TABLE • *102*, *248*, *338*, *343*, 344, 358, *361*, *401*, 409, 411, 417, *419*, 420, 434, 498, 506, 519, 531, *649*, 651, *832*, *833*, *834*, *835*, *836*, *841*, *843*, *845*, *846*, *848*, *850*, *851*, *852*, *854*, *856*, *858*, *859*, *860*, *862*, *863*, *865*, *866*, *867*, *868*, *870*, *872*, *874*, *876*, *878*, *886*, *889*, *901*, 936
<table commit action> • *361*, **362**
<table constraint> • 375, 376, ***384***, 385, 409, 434, 437, 743
<table constraint definition> • *363*, 364, 365, 366, 375, 376, **384**, 409, *416*, 434, 437, *485*, 486, 615, 622, 853
table constraint descriptor • 49, 50, 384, 385, 390, 410, 416, 516, 519
<table definition> • 45, 46, 47, 53, 69, 119, 224, 226, *355*, **361**, 363, 364, 365, 366, 367, 368, 369, 370, 373, 375, 376, 384, 386, 387, 389, 398, 399, 404, 466, 497, 498, *553*, 594, 595, 596, 624, 629, 632, 639, 644, 652, 741, 830, 958
table descriptor • 17, 47, 48, 59, 91, 223, 363, 369, 377, 387, 390, 402, 408, 410, 413, 416, 417, 421, 437, 517, 615, 845, 849
<table element> • ***363***, 364, 365, 366, 369, 370, 371, 917
<table element list> • *362*, **363**, 364, 365, 366, *649*
<table expression> • 55, 144, 147, 149, **217**, 218, 231, *237*, 238, 240, 241, 242, 246, 269, 611, *623*, 624, 632, 924, 927, 933, 934
<table name> • 14, 59, **115**, 119, 120, 122, *143*, 144, 145, *146*, 147, 220, 239, 240, 242, 245, *248*, 249, 252, 253, 307, *343*, 344, 358, *361*, *363*, 365, 366, 367, 368, 369, 370, 375, 386, 387, *389*, 390, 391, 395, 399, *401*, 402, 403, 408, 410, 411, 416, 417, *419*, 420, *421*, 422, 423, 424, *426*, 434, *459*, 460, *464*, 466, 467, 468, 470, 471, 472, 473, 477, 478, 498, 504, 505, 512, 513, 515, 519, 585, 596, *625*, 626, 628, 632, 633, 640, 642, 644, 645, 649, *652*, 653, 664, 717, *729*, *730*, *732*, *733*, 741, 743, 749, 882, 921, 924, 928, 930
<table of existing type> • *361*, **362**, 364, 370
<table of new type> • *361*, **362**, 364, 366
<table reference> • 53, 54, **143**, 144, 145, 146, 147, 149, *219*, 220, *221*, 225, 231, 239, 240, 241, 250, 251, 252, *307*, 398, 421, 422, 470, 472, 473, 505, 513, 515, 516, 524, 525, *628*, 629, *631*, 632, *638*, 639, *643*, 644, 923, 924, 932
<table reference list> • ***307***, *309*, 525
TABLES • 797, 798, 845, 846, 848, 850, 852, 859, 863, 868, 901, 931
<table scope> • ***361***
<table subquery> • *143*, 144, *184*, 186, **268**, *280*, *298*, 299, *300*, *301*, *302*, 303, 304, 524, 705, 706
<table type> • *203*, 204, 220, 232, 233, *248*, *343*, 344, ***361***, 364, 369, *401*, *419*, 531, 532, 533, 632, 634, 643, *649*
<table value constructor> • 53, 209, **210**, *248*, 250, 252, 258, 280, 637, 704, 705, 924, 933, 937
<table value constructor list> • ***210***
<table value expression> • 53, *184*, 185, **203**, 204, 323, 531, 532, 533
TABLE_CATALOG • 797, 798, 799, 800, 804, 805, 808, 810, 813, 818, 819, 820, 821, 822, 839, 845, 846, 848, 849, 850, 851, 852, 853, 854, 859, 860, 863, 864, 868, 869, 870, 871, 891, 892, 893, 894, 901
TABLE_CONSTRAINTS • 810, 820, 821, 852, 853, 854, 856, 858, 898, 900
TABLE_NAME • *100*, *735*, 737, 742, 743, *797*, *798*, *799*, *800*, *804*, *805*, *808*, *810*, *813*, *818*, *819*, *820*, *821*, *822*, *839*, *845*, *846*, *848*, *849*, *850*, *851*, *852*, *853*, *854*, *859*, *860*, *863*, 864, *868*, 869, *870*, 871, *891*, *892*, *893*, *894*, *901*
TABLE_OR_DOMAIN_CATALOG • 794, 800, 836, 838, 839, 841, 848, 849
TABLE_OR_DOMAIN_NAME • 794, 800, 836, 838, 839, 841, 848, 849
TABLE_OR_DOMAIN_SCHEMA • 794, 800, 836, 838, 839, 841, 848, 849
TABLE_PRIVILEGES • 797, 804, 863, 868, 891, 931
TABLE_SCHEMA • 797, 798, 799, 800, 804, 805, 808, 810, 813, 818, 819, 820, 821, 822, 839, 845, 846, 848, 849, 850, 851, 852, 853, 854, 859, 860, 863, 864, 868, 869, 870, 871, 891, 892, 893, 894, 901
TABLE_TYPE • 797, 845, 846, 852
<target character set specification> • ***454***, 455, 514
<target specification> • 13, 80, 121, 133, **134**, 137, 336, 340, 398, 421, 459, 468, 524, 549, 578, *592*, 614, *617*, 618, 620, *623*, 624, *631*, 636, 697, 700, *710*, 714, 715, 718, 726, 783, 937, 955
TARGET_CHARACTER_SET_CATALOG • 817, 878, 879
TARGET_CHARACTER_SET_NAME • 817, 878, 879
TARGET_CHARACTER_SET_SCHEMA • 817, 878, 879
TEMPLATE • *102*, 341, *343*, 359, *495*, *497*, *522*, *526*, *528*
<template parameter> • ***341***

<template parameter declaration> • 121, ***495***
<template parameter declaration list> • 341, ***495***, 496
<template parameter list> • ***341***
<template parameter name> • **117**, 121, *125*, 131, *133*, *134*, 341, *495*
<template parameter type> • 37, 121, ***495***
TEMPORARY • 45, *102*, *361*, 366, 371, *649*, *652*, *845*, *889*, *890*, *891*, *892*, *893*, *894*, *895*, *896*, *897*, 925
temporary table • 45, 46, 47, 61, 70, 74, 75, 76, 78, 89, 90, 93, 119, 120, 122, 307, 344, 358, 361, 369, 371, 373, 376, 390, 398, 401, 419, 421, 459, 538, 585, 586, 626, 628, 633, 640, 644, 649, 650, 651, 652, 664, 681, 717, 742, 743, 749, 752, 753, 845, 889, 920, 925, 927, 928, 930, 931, 941, 945, 956, 959
<temporary table declaration> • 46, 61, 70, 74, 75, 76, 78, 119, 373, 376, 398, *535*, 538, *584*, 585, **649**, 650, 651, 652, *747*, 749, 752, 753, *783*, 920, 927, 928, 930, 931, 941
temporary view • 45, 47, 49, 89, 90, 119, 147, 307, 358, 421, 459, 538, 585, 586, 650, 652, 653, 681, 749, 752, 753, 754, 890, 891, 892, 893, 894, 895, 896, 897, 922, 956, 960
<temporary view definition> • 47, 89, 147, *535*, 538, *584*, 650, **652**, 653, *747*, 749, 752, 753, 754, 922
<term> • ***187***, *199*, 200
<terminated SQL statement> • ***584***
TEST • *102*, *659*, 741, 967
<test completion statement> • *554*, 556, 571, 577, 579, **659**, 660, 664, 741, 920, 922
THAN • *102*, *485*, *531*, *532*, *533*, *801*, *835*, *900*
THEN • *102*, *165*, 166, *446*, 449, 450, 585, *587*, *597*, *599*, *798*, *799*
THERE • *102*, *309*, 967
<there is clause> • 231, ***309***
<there is predicate> • 147, *271*, 272, **309**, 916
TIME • 21, 22, 30, 31, 32, 66, *102*, *107*, 111, *126*, 129, 130, 147, 148, 153, 162, 180, 181, *196*, 197, 557, 559, 561, 564, 566, 567, 569, *684*, 690, 691, 701, 705, *837*
<time fractional seconds precision> • 31, ***126***, 129, 130, 153, 557, 947
<time interval> • ***108***
<time literal> • ***107***, 111, 113
<time precision> • ***126***, 128, 129, 130, 132, *162*, 163, 688, 711, 713, 923, 924, 947
TIMESTAMP • 21, 22, 30, 31, 32, 66, *102*, *107*, 111, *126*, 129, 130, 147, 148, 153, 162, 181, 197, 557, 559, 561, 564, 566, 567, 569, 690, 691, 701, 705, *837*
<timestamp literal> • ***107***, 111, 113
<timestamp precision> • ***126***, 128, 129, 130, 132, *162*, 163, 688, 711, 713, 923, 924, 947
<timestamp string> • *98*, 103, ***107***
<time string> • *98*, ***107***
<time value> • *107*, **108**, 113
<time zone> • ***196***, 197, 198
<time zone field> • ***152***, 153
<time zone interval> • ***107***, 111, 113
<time zone specifier> • ***196***, 197, 198
TIMEZONE_HOUR • 31, *102*, 130, *152*, 154
TIMEZONE_MINUTE • 31, *102*, 130, *152*
TO • *32*, *102*, 148, 158, 159, 182, 196, 201, *260*, 275, *286*, 289, 291, 293, 294, 295, 324, *327*, 348, 443, 448, *454*, 455, *503*, 504, 505, *508*, 571, *666*, *669*, 684, 691, 701, *755*, 756, 757, 758, 759, 785, *838*
<token> • **98**, 103, 749
too many • 661, 907
TRAILING • *102*, *155*, 160
transaction • 10, 49, 58, 59, 65, 66, 68, 69, 71, 73, 74, 76, 77, 78, 85, 86, 87, 88, 89, 91, 92, 124, 353, 392, 395, 396, 425, 461, 466, 477, 553, 554, 556, 571, 572, 578, 614, 615, 618, 621, 622, 626, 627, 628, 630, 633, 636, 640, 642, 644, 647, 655, 656, 657, 661, 663, 664, 665, 666, 667, 669, 672, 674, 681, 683, 702, 707, 741, 742, 754, 783, 784, 785, 786, 904, 906, 907, 908, 920, 927, 929, 931, 937, 938, 943, 944, 951, 953, 956, 960
TRANSACTION • *102*, *547*, *548*, *655*, 741
<transaction access mode> • ***655***
transaction-initiating • 85, 572, 669, 672, 784, 785
<transaction mode> • ***655***
transaction resolution unknown • 89, 904
TRANSLATE • *102*, *155*
translation • 10, 23, 24, 59, 70, 82, 83, 93, 117, 122, 123, 124, 155, 156, 158, 159, 161, 343, 344, 355, 356, 357, 358, 359, 428, 445, 446, 448, 449, 450, 452, 454, 455, 456, 457, 504, 505, 512, 513, 514, 516, 517, 520, 553, 554, 704, 739, 741, 817, 873, 878, 879, 919, 923, 925, 926, 927, 929, 931, 948, 949
TRANSLATION • *102*, *343*, 359, *446*, *454*, 455, *457*, 505, 506, 739, 741, *817*, *872*, 873, *878*, 879, 927
<translation basis> • *454*, **455**, 456
<translation collation> • 23, ***446***, 448, 450, 457
<translation definition> • 23, 70, *355*, 357, **454**, *455*, 456, 505, 514, *553*, 741, 925, 926, 927, 929, 931
translation descriptor • 24, 59, 455, 456
<translation modifier> • ***455***
<translation modifiers> • ***454***
<translation name> • 23, 24, **117**, 122, 123, 124, *155*, 156, 158, *343*, 344, 359, 428, *446*, 448, 449, 450, *454*, 455, *457*, 504, 512, 514, 923
<translation option> • ***454***
<translation options> • ***454***, 455
<translation routine> • ***454***, 456, 919
TRANSLATIONS • 817, 872, 878
<translation source> • ***454***, 456, 919
<translation specification> • ***454***, 456, 919
TRANSLATION_CATALOG • 817, 872, 878, 879
TRANSLATION_DEFINITION • 879
TRANSLATION_NAME • 817, 872, 878, 879
TRANSLATION_SCHEMA • 817, 872, 878, 879

TRIGGER • 82, 84, *102*, *343*, 347, 359, 410, *464*, 472, *478*, 504, 519, *548*, *735*, 737, 739, 741, 742, 743, *824*, *825*, *826*, *863*, 864, *868*, 869, *881*, 882, *883*, *885*, 917, 968
trigger action graph • 470, 471, 472
<trigger action time> • 58, 144, 239, ***464***, 467, 469, 473, 475, 476, 478, 882
<trigger column list> • 403, 411, ***465***, 466, 469, 470, 471, 473, 478, 882
<trigger definition> • 16, 58, 70, 121, 144, 147, 239, *247*, 254, *355*, 356, 410, **464**, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, *553*, 556, 741, 917, 919, 920
trigger descriptor • 58, 391, 403, 411, 426, 469, 473, 478, 481, 517, 519, 881
<triggered action> • 403, *464*, **465**, 468, 469, 473, 474, 475, 476, 477, 629, 645, 884
<triggered assertion> • ***459***, 460, 461, 919
triggered data change violation • 394, 395, 396, 477, 742, 908
<triggered SQL statement> • 16, 58, 86, ***465***, 466, 467, 468, 470, 471, 472, 474, 475, 476, 477, 517, 628, 632, 633, 644, 665
<triggered SQL statement list> • ***465***, 470, 471, 476, 477, 884
<trigger event> • 144, 239, 391, *464*, **465**, 467, 468, 469, 470, 471, 472, 473, 475, 476, 477, 478, 882
<trigger name> • **117**, 123, 359, 410, *464*, 466, 469, 470, 472, 473, *478*, 519, 743
TRIM • *102*, *155*, 157, 172, 180, 181, 182, *546*, 669, 670, 677, 678, 680, 681, 692, 706, 722
<trim character> • ***155***, 157, 160, 704
<trim function> • 24, ***155***, 157, 160, 161, 704, 932
<trim operands> • ***155***
<trim source> • ***155***, 157, 160, 704
<trim specification> • ***155***, 157, 160
TRUE • *102*, *108*, 177, 179, 262, 264, *313*
<truth value> • ***313***, 315, 925
TUPLE • *102*, *582*
<tuple constructor> • ***582***, 583
TYPE • *100*, *102*, 121, 136, 342, *343*, 359, *362*, 364, 368, 370, 376, *483*, *493*, *495*, *497*, *498*, 520, *522*, *523*, *526*, *528*, *529*, *531*, *532*, *533*, 541, *545*, *546*, *547*, *548*, 557, 687, 688, 689, *695*, 696, 697, 700, 701, 711, 712, 713, 714, 715, 716, 739, 741, *760*, 762, *794*, *797*, *799*, *800*, *801*, *802*, *803*, *804*, *805*, *806*, *807*, *808*, *809*, *810*, *815*, *816*, *817*, *835*, *836*, *837*, *838*, *839*, 840, *841*, *845*, *846*, *848*, *849*, *852*, *853*, *854*, *856*, *861*, *863*, 864, *868*, 869, *870*, 871, *872*, 873, *876*, 877, *891*, *892*, *893*, *894*, *900*, 952, 958, 960, 968
typeset • 333, 334
type template • 9, 10, 37, 38, 70, 82, 83, 121, 124, 136, 341, 342, 344, 356, 358, 359, 487, 495, 496, 497, 512, 518, 520, 521, 522, 526, 528, 529, 541, 556, 920
<type template definition> • 70, 121, 136, 341, *355*, 356, 487, **495**, 496, 497, 518, 521, 541, *553*, 556, 920
type template family • 124, 341, 495
<type template name> • **117**, 124, *341*, 342, *343*, 344, 359, *495*, 496, *497*, 512, 520

— U —

UNCOMMITTED • 86, 87, *100*, *655*
UNDER • 30, 37, 82, 84, *102*, 177, 179, 205, 206, 264, 266, 296, 313, *343*, 344, 347, *362*, 369, 484, *485*, 489, 492, 518, *526*, *528*, *532*, *533*, 757, 917, 968
underlying • 45, 147, 168, 185, 217, 238, 240, 241, 245, 252, 253, 422, 423, 424, 470, 473, 504, 505, 513, 515, 610, 625, 630, 632, 634, 637, 639, 647, 653, 729, 730, 732, 733
<underscore> • ***96***, *98*, 104, 105, *106*, *115*, 118, 283, *287*, 288, 289, 290, 292, 443, 922
UNION • 52, 53, 56, *102*, 210, *221*, 222, 223, 226, 227, 229, 230, *247*, 249, 252, 254, 255, 257, 259, *260*, *265*, 275, 632, *797*, *820*, *821*, *839*, *853*, *858*, *872*, *890*, *891*, *892*, *893*, *894*, *895*, *896*, *898*, 924, 934
UNIQUE • 50, *102*, *301*, *302*, 303, 304, 364, 366, 384, *386*, 387, 388, *811*, *820*, *848*, *852*, *853*, *854*, *856*, 857, *867*, *898*, *899*, *900*, 935
<unique column list> • 50, 53, 223, 224, 226, 384, ***386***, 387, 388, 390, 918, 935
<unique constraint definition> • 226, *384*, **386**, 387, 388, 390, 918
unique matching rows • 86, 392, 393, 394
<unique predicate> • *271*, 272, **301**, 492, 524, 934
<unique specification> • 223, 366, 368, *373*, 375, ***386***, 387
UNIQUE_CONSTRAINT_NAME • 811, 820, 856, 857, 898, 899
<universal clause> • 147, ***307***, 308, 525
UNKNOWN • *102*, *313*, *545*
UNNAMED • *100*, 689, *695*, 697, 699, 711, 712
<unparenthesized style list> • ***539***
<unqualified schema name> • ***115***, 120, 121, 356, 678, 742, 743, 744, 789
unrepresentable value for USER • 137, 905
<unsigned integer> • ***107***, *108*, 112, 113, *126*, 176, 178, 261, *327*, 610, *755*, 756, *757*, 758, 759, 787, 923, 963
<unsigned literal> • **106**, *133*, 923, 931
<unsigned numeric literal> • *98*, *106*, ***107***, *442*
<unsigned value specification> • **133**, 135, 136, 137, *184*, 185
<updatability clause> • ***609***, 610, 611, 613, 639, 721, 730, 927, 930, 937
updatable • 43, 45, 48, 67, 144, 226, 239, 240, 241, 245, 246, 247, 249, 362, 370, 372, 377, 421, 423, 424, 473, 478, 485, 504, 513, 590, 610, 611, 612, 614, 625, 632, 633, 634, 638, 639, 643, 644, 652, 653, 724, 729, 730, 847, 916, 933

UPDATE • 58, 82, 83, *102*, 144, *237*, 239, *343*, 344, 346, 370, *389*, 403, 404, 411, 423, 424, *459*, 461, *465*, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 503, 504, 512, 513, 521, *609*, 610, 611, 613, 615, 626, 628, 633, *638*, 639, 640, *643*, 644, 651, 653, 721, *730*, *733*, 740, 741, *811*, *856*, 857, *863*, 864, *868*, 869, *870*, 871, *881*, 882, 927, 930
<update by moving> • ***643***, 646
<update by setting> • ***643***, 646
<update mechanism> • ***643***
update operation • 424
<update rule> • ***389***, 390, 393, 394, 396, 397, 857, 925
<update statement: positioned> • 67, 71, 74, 75, 77, 78, 245, 257, 392, 395, 396, 424, 425, 461, 475, *554*, 610, 626, 629, **638**, 639, 640, 641, 642, 645, 730, 731, 733, 741, 754, 921, 938
<update statement: searched> • 67, 71, 74, 75, 76, 78, 144, 147, *237*, 244, 245, 257, 381, 395, 396, 424, 425, 461, *465*, 467, 468, 470, 471, 472, 475, 477, 524, *554*, 625, 626, 641, **643**, 644, 647, 648, *702*, 741, *783*, 922, 938, 960
<update target> • ***638***, 642, 647, 921, 922
<update type> • ***638***, 642, *643*, 647, 921, 922
UPDATE_RULE • 811, 856, 857
UPPER • *102*, *155*, 159, 287, 295
USAGE • 82, 83, *102*, 129, 158, 170, 342, *343*, 344, 348, 351, 369, 376, 409, 428, 437, 438, 439, 441, 443, 444, 445, 448, 449, 450, 452, 455, 456, 457, 483, 494, 496, 503, 504, 505, 512, 514, 516, 517, 518, 561, *767*, 768, 769, 789, *794*, *802*, *803*, *806*, *813*, *815*, *816*, *817*, *818*, *819*, *820*, *821*, *822*, *850*, *851*, *854*, *859*, *860*, *861*, *872*, *899*, *900*, 931, 949, 952
USAGE_PRIVILEGES • 794, 802, 803, 806, 815, 816, 817, 872, 931
USER • 87, 93, *100*, *102*, *133*, 135, 137, 138, 239, *379*, 380, 381, 382, 383, 398, 459, 615, 640, *669*, 787, *793*, *794*, *796*, *797*, *798*, *799*, *800*, *801*, *802*, *803*, *804*, *805*, *806*, *807*, *810*, *811*, *812*, *813*, *814*, *815*, *816*, *817*, *818*, *819*, *820*, *821*, *822*, *823*, *824*, *825*, *826*, *832*, *833*, *834*, *838*, *861*, *863*, *865*, *868*, *870*, *872*, *889*, *890*, *891*, *892*, *893*, *894*, *895*, *896*, *897*, 905, 932, 935, 947, 963
<user-defined character repertoire name> • ***348***, 349
<user-defined delete clause> • ***237***, 244, 256, 257
<user-defined insert clause> • ***237***, 244, 245, 256, 257
<user-defined updatability clause> • ***237***, 242, 244, 245, 246, *247*, 249, 253, 256, 257, 258, 916
<user-defined updatability method> • ***237***
<user-defined update clause> • ***237***, 244, 245, 256, 257
<user name> • 90, **118**, 122, *669*, 670, 951
USERS • 832, 833, 834, 861, 863, 865, 868, 870, 872, 889
USER_NAME • 832, 833, 863, 865, 889, 890, 891, 892, 897
USER_ROLE_GRANTS • 895
USER_SESSION_ENABLED_ROLES • 889, 890
USING • *102*, *155*, *221*, *447*, 448, 449, *454*, 455, *546*, *547*, *710*, *794*, *821*, *900*
<using arguments> • ***710***, 713, 714, 717, 726
<using clause> • **710**, 713, 714, 715, *717*, *724*, *726*
using clause does not match dynamic parameter specifications • 713, 714, 906
using clause does not match target specifications • 714, 715, 906
using clause required for dynamic parameters • 718, 724, 906
using clause required for result fields • 718, 906
<using descriptor> • *709*, ***710***, 713, 714, 715, 717, 718, 726
UTC • 7, 32, 113, 148, 198

— V —

<val> • ***442***
VALUE • 43, *102*, *133*, 135, 136, 138, 409, 428, 434, 437, *526*, *589*, *695*, *699*, 768, *829*, 932, 952
<value expression> • 53, 54, 55, 56, 62, 141, 144, 148, *149*, 150, 151, *165*, 166, *168*, 170, 171, ***184***, 185, 186, *194*, *203*, 204, *207*, 210, *211*, 222, 225, 231, 232, 234, *237*, 238, 239, 241, 243, *248*, 249, *260*, *280*, 281, 296, *333*, 335, *582*, 583, *592*, 593, *597*, *609*, 611, 612, 623, 639, 641, 642, 643, 644, 646, 647, 648, 703, 704, 705, 706, 730, 915, 923, 925, 928, 932, 933, 938
<value expression list> • ***582***, 924
<value expression primary> • ***184***, 185, 186, *187*, *190*, 191, *194*, *196*, *199*, *205*, 933
VALUES • *102*, *210*, 280, *338*, *631*, 632, *843*, *846*, *852*, *862*, 937
<value specification> • 121, **133**, 135, 136, 137, *141*, 149, 150, *164*, *260*, 262, 264, 281, 285, 336, *341*, 398, 459, 468, 549, 592, 615, 628, 633, 637, 644, *677*, *678*, *680*, *681*, 704, 706, 783, 787, 925, 933, 934, 947, 952
VARCHAR • *102*, *125*, 127, 132, *763*, 764, 765, 766, *774*, 775, 931, 940
variable • 7, 9, 16, 25, 28, 41, 64, 65, 66, 77, 79, 80, 81, 87, 92, 117, 123, 127, 133, 134, 135, 136, 137, 138, 139, 140, 156, 157, 169, 172, 173, 176, 178, 179, 180, 190, 191, 192, 193, 239, 264, 266, 284, 293, 317, 318, 320, 321, 323, 338, 379, 380, 382, 383, 524, 535, 536, 537, 538, 576, 584, 585, 586, 590, 591, 592, 594, 595, 604, 702, 703, 710, 719, 747, 748, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 771, 772, 774, 775, 776, 777, 779, 780, 781, 783, 828, 920, 923, 930, 940, 944, 946, 947, 950, 952, 958, 959, 961
VARIABLE • *102*, 968
variable-length coding • 7, 9
<variable specification> • *133*, ***134***, 136, 137, 536
VARIANT • *102*, 339, *540*, 549, 552
<variant attribute> • ***540***, 549

VARYING • 21, 22, 27, 40, *102*, *125*, 127, 132, 375, 427, *522*, *523*, *528*, 557, 559, 561, 564, 566, 567, 569, 573, 574, 575, 576, 577, 687, 688, 690, 700, 704, 706, 716, 764, 765, 774, *779*, 780, 781, *828*, *836*, 927, 931, 937, 940
<vertical bar> • ***96***, *286*, 288, 289, 290, 292
view • 13, 14, 42, 45, 47, 48, 49, 59, 60, 69, 82, 84, 85, 89, 90, 91, 92, 94, 119, 144, 147, 218, 220, 239, 242, 245, 246, 269, 307, 355, 356, 358, 403, 410, 411, 419, 421, 422, 424, 425, 426, 436, 438, 445, 457, 459, 460, 470, 473, 481, 497, 498, 504, 505, 513, 514, 515, 516, 518, 519, 535, 538, 553, 584, 585, 586, 624, 634, 636, 641, 646, 650, 651, 652, 653, 681, 724, 739, 741, 743, 747, 749, 752, 753, 754, 789, 793, 794, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 830, 845, 846, 847, 849, 850, 851, 890, 891, 892, 893, 894, 895, 896, 897, 922, 924, 926, 927, 934, 936, 941, 955, 956, 959, 960, 969
VIEW • *102*, 358, 411, *421*, *426*, 498, 519, *652*, 653, 739, 741, *793*, *794*, *796*, *797*, *798*, *799*, *801*, *802*, *803*, *804*, *805*, *806*, *807*, *808*, *809*, *810*, *811*, *812*, *813*, *814*, *815*, *816*, *817*, *818*, *819*, *820*, *821*, *822*, *823*, *824*, *825*, *826*, *827*, *845*, *846*, *850*, *851*, *852*, *890*, *891*, *892*, *893*, *894*, *895*, *896*, *897*, 931
<view column list> • ***421***, 422, 513, 514, 515, 516, *652*, 653
<view definition> • 14, 45, 69, 119, 144, 147, 239, 242, 245, *355*, **421**, 422, 425, 457, 473, 497, 498, 513, 514, 515, 516, *553*, 652, 741, 830, 846, 847, 936
view descriptor • 47, 48, 59, 403, 410, 419, 422, 424, 426, 445, 457, 481, 516, 519, 653, 846, 849
viewed table • 13, 45, 47, 85, 92, 421, 422, 426, 652, 653, 789, 798, 818, 819, 826, 845, 846, 956, 960
VIEWS • 798, 845, 846, 850, 851, 852, 931
VIEW_CATALOG • 818, 819, 850, 851
VIEW_COLUMN_USAGE • 819, 851, 931
VIEW_DEFINITION • 798, 846
VIEW_NAME • 818, 819, 850, 851
VIEW_SCHEMA • 818, 819, 850, 851
VIEW_TABLE_USAGE • 818, 850, 931
VIRTUAL • *102*, *372*, 968
<virtual column> • 366, ***372***, 374, 375, 377
visible • 8, 9, 35, 36, 68, 88, 241, 253, 474, 475, 616, 626, 629, 633, 634, 640, 645, 951, 956
VISIBLE • 35, 36, *102*, 170, *362*, 365, 487, 544, 968

— W —

WAIT • *102*, *659*, 664, 968
warning • 64, 85, 150, 176, 177, 179, 180, 318, 339, 381, 396, 399, 425, 461, 505, 520, 578, 626, 629, 635, 641, 645, 675, 710, 712, 742, 786, 903, 908
WHEN • *102*, 165, 166, *447*, 448, 449, *465*, 585, *587*, *597*, *798*, *799*
WHENEVER • *102*, *755*
<when operand> • ***165***, 166, 703, 705
WHERE • 54, 55, 56, *102*, *231*, 385, 387, 428, *625*, *628*, *638*, *643*, *729*, *730*, *732*, *733*, 739, 741, *793*, *794*, *796*, *797*, *798*, *799*, *800*, *801*, *802*, *803*, *804*, *805*, *806*, *807*, *808*, *809*, *810*, *811*, *812*, *813*, *814*, *815*, *816*, *817*, *818*, *819*, *820*, *821*, *822*, *823*, *824*, *825*, *826*, *845*, *846*, *852*, *853*, *854*, *856*, *890*, *891*, *892*, *893*, *894*, *895*, *896*, *897*, *899*, *900*
<where clause> • 54, 143, *217*, 218, **231**, 232, 233, 235, 241, 246, *309*, *631*, 634, 646, 924, 933
WHILE • *102*, *602*, *605*, 968
WITH • 30, 31, 32, 35, 36, 41, 82, 83, 91, 92, *102*, *126*, 129, 130, 147, 148, 153, 162, 164, 170, 181, 245, 346, 348, *362*, 365, 366, *372*, *421*, 422, 424, 425, 484, *485*, *486*, 487, 489, *503*, 504, 505, 507, *508*, 510, 511, 514, 515, *522*, *526*, *528*, 544, 636, 641, 646, 691, *692*, 701, 705, 743, 789, *837*, 847, 864, 865, 869, 871, 873, 895, 896, 951
WITH CHECK OPTION • 92, 422, 424, 425, 636, 641, 646, 847
with check option violation • 424, 743, 909
WITH GRANT OPTION • 83, 91, 504, 505, 514, 515, 789, 864, 869, 871, 873
WITHOUT • 36, *102*, *362*, 365, 487, *531*, *532*, *533*, 968
WORK • *102*, *664*, 665, *666*, 667, 739, 740, 938
WRITE • *102*, *655*, 656

— Y —

YEAR • 31, 32, 33, *102*, 111, 129, 130, 324, *327*, 328, 691, 705, *827*, *838*, *886*, 887
<year-month literal> • ***108***, 111, 329
<years value> • ***108***, 111, 329

— Z —

ZONE • 30, 31, 32, *102*, 111, *126*, 129, 130, 147, 148, 153, 162, 181, *196*, *684*, 691, 701, 705, 740, *837*