

STANDARDS AUSTRALIA/STANDARDS NEW ZEALAND

Amendment No. 1
to
AS/NZS 3968:1994
Information technology — Database languages — SQL

CORRECTION

The 1994 edition of AS/NZS 3968 is amended as follows; the amendments should be inserted in the appropriate place.

SUMMARY: This Amendment applies to the title of the Standard and to pages 283 to 286 and pages 431 to 434, which were inadvertently reversed in the printing process. The Table of Contents is correct.

Published on 5 December 1996.

AMDT
No. 1
DEC.
1996

Title page

Delete title and insert the following:

Information technology—Database
languages—SQL

Part 0: Definition of data structures and basic
operations

AMDT
No. 1
DEC.
1996

Pages 283 and 285

Change number of page 283 to 285 and change number of page 285 to 283.

AMDT
No. 1
DEC.
1996

Pages 284 and 286

Change number of page 284 to 286 and change number of page 286 to 284.

AMDT
No. 1
DEC.
1996

Pages 431 and 433

Change number of page 431 to 433 and change number of page 433 to 431.

AMDT
No. 1
DEC.
1996

Pages 432 and 434

Change number of page 432 to 434 and change number of page 434 to 432.

AS/NZS 3968:1994
ISO/IEC 9075:1992

Australian/New Zealand Standard

**Information technology—Database
languages—SQL**

STANDARDS AUSTRALIA 

STANDARDS NEW ZEALAND

 **Documents**
Document Sharing Hub

AS/NZS 3968:1994

This Joint Australian/New Zealand Standard was prepared by Joint Technical Committee IT/1, Information Systems—Interconnection. It was approved on behalf of the Council of Standards Australia on 19 May 1994 and on behalf of the Council of Standards New Zealand on 9 May 1994. It was published on 18 July 1994.

The following interests are represented on Committee IT/1:

Australian Bankers Association
 Australian Bureau of Statistics
 Australian Chamber of Commerce and Industry
 Australian Committee of Directors and Principals
 Australian Computer Society
 Australian Computer Users Association
 Australian Information Industry Association
 Australian Vice Chancellors Committee
 Department of Defence, Australia
 Department of Industry, Technology and Regional Development, Australia
 Government Computing Service, New Zealand
 Information Exchange Steering Committee, Australia
 Institute of Information and Communication Technologies, CSIRO, Australia
 OPTUS Communications, Australia
 Telecom Australia
 Telecom New Zealand

Review of Standards. To keep abreast of progress in industry, Joint Australian/New Zealand Standards are subject to periodic review and are kept up to date by the issue of amendments or new editions as necessary. It is important therefore that Standards users ensure that they are in possession of the latest edition, and any amendments thereto.

Full details of all Joint Standards and related publications will be found in the Standards Australia and Standards New Zealand Catalogue of Publications; this information is supplemented each month by the magazines 'The Australian Standard' and 'Standards New Zealand', which subscribing members receive, and which give details of new publications, new editions and amendments, and of withdrawn Standards.

Suggestions for improvements to Joint Standards, addressed to the head office of either Standards Australia or Standards New Zealand, are welcomed. Notification of any inaccuracy or ambiguity found in a Joint Australian/New Zealand Standard should be made without delay in order that the matter may be investigated and appropriate action taken.

AS/NZS 3968:1994

Australian/New Zealand Standard

Information technology—Database languages—SQL

First published as AS 3698—1991.
Jointly revised and designated as Joint Standard
AS/NZS 3968:1994.

PUBLISHED JOINTLY BY:

STANDARDS AUSTRALIA
1 The Crescent,
Homebush NSW 2140 Australia

STANDARDS NEW ZEALAND
Level 10, Standards House,
155 The Terrace,
Wellington 6001 New Zealand

PREFACE

This Standard was prepared by the Joint Standards Australia/Standards New Zealand Committee IT/1 on Information Systems—Interconnection to supersede AS 3968, *Information processing systems—Database language SQL with data enhancement*. It is identical with and has been reproduced from ISO/IEC 9075:1992, *Information technology—Database languages—SQL*.

This Standard is issued as a Joint Standard under the terms of the Active Cooperation Agreement between Standards Australia and Standards New Zealand.

The objective of this Standard is to provide users of large databases with a definition of data structures and basic operations performed on SQL-data and to provide functional capabilities for creating, accessing, maintaining, controlling and protecting SQL-data.

This Standard is one of a series of Open Systems Interconnection (OSI) Standards which are currently under development. Since OSI Standards are developmental, there may be some minor difficulties encountered in their implementation. For this reason, Standards Australia will be providing, through the OSI Help Desk, a service to coordinate and disseminate information concerning difficulties which are identified in using this Standard.

Users of this Standard are advised by Standards Australia and Standards New Zealand, under arrangements made with ISO and IEC, as well as certain other Standards organizations, that the number of this Standard is not reproduced on each page; its identity is shown only on the cover and title pages.

For the purpose of this Standard, the source text should be modified as follows:

- (a) *Terminology* The words 'this Australian/New Zealand Standard' should replace the words 'this International Standard' wherever they appear.
- (b) *Decimal marker* Substitute a full point for a comma where it appears as a decimal marker.
- (c) *References* The references to international Standards should be replaced by references, where appropriate, to the following Australian or Joint Australian/New Zealand Standards:

<i>Reference to International Standard or other publication</i>		<i>Australian or Joint Australian/New Zealand Standard</i>	
ISO		AS	
1989	Programming languages—COBOL	1209	Programming language COBOL
2022	Information technology—ISO 7-bit and 8-bit coded character sets—code extension techniques	1953	Information processing—ISO 7-bit and 8-bit coded character sets—Code extension techniques
6160	Programming languages—PL/I	—	
7185	Information technology—Programming languages—Pascal	2580	Programming languages—Pascal
8601	Data elements and interchange formats—Information interchange—Representation of dates and times	3802	Data elements and interchange formats—Information interchange—Representation of dates and times
8652	Programming languages—Ada	—	
ISO/IEC			
646	Information technology—ISO 7-bit coded character set for information interchange	1776	Information processing—7-bit coded character set for information interchange
		AS/NZS	
1539	Information technology—Programming languages—Fortran	1486	Information technology—Programming languages—Fortran 90



Documents

Document Sharing Hub

iii

ISO/IEC	AS
8824 Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1)	3625 Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1)
9579 Information technology—Open Systems Interconnection—Remote Database Access	
9579-2 Part 2: SQL specialization	—
9899 Programming languages—C	3955 Programming languages—C
10206 Information technology—Programming languages—Extended Pascal	3981 Information technology—Programming languages—Extended Pascal
10646 Information technology—Multiple-octet coded character set	—
11756 Information technology—Programming languages—MUMPS	—

© Copyright – STANDARDS AUSTRALIA/STANDARDS NEW ZEALAND

Users of Standards are reminded that copyright subsists in all Standards Australia and Standards New Zealand publications and software. Except where the Copyright Act allows and except where provided for below no publications or software produced by Standards Australia or Standards New Zealand may be reproduced, stored in a retrieval system in any form or transmitted by any means without prior permission in writing from Standards Australia or Standards New Zealand. Permission may be conditional on an appropriate royalty payment. Australian requests for permission and information on commercial software royalties should be directed to the head office of Standards Australia. New Zealand requests should be directed to Standards New Zealand.

Up to 10 percent of the technical content pages of a Standard may be copied for use exclusively in-house by purchasers of the Standard without payment of a royalty or advice to Standards Australia or Standards New Zealand.

Inclusion of copyright material in computer software programs is also permitted without royalty payment provided such programs are used exclusively in-house by the creators of the programs.

Care should be taken to ensure that material used is from the current edition of the Standard and that it is updated whenever the Standard is amended or revised. The number and date of the Standard should therefore be clearly identified.

The use of material in print form or in computer software programs to be used commercially, with or without payment, or in commercial contracts is subject to the payment of a royalty. This policy may be varied by Standards Australia or Standards New Zealand at any time.

CONTENTS

	<i>Page</i>
1 Scope	1
2 Normative references	3
3 Definitions, notations, and conventions	5
3.1 Definitions	5
3.1.1 Definitions taken from ISO/IEC 10646	5
3.1.2 Definitions taken from ISO 8601	5
3.1.3 Definitions provided in this International Standard	5
3.2 Notation	7
3.3 Conventions	8
3.3.1 Informative elements	8
3.3.2 Specification of syntactic elements	8
3.3.3 Specification of the Information Schema	9
3.3.4 Use of terms	9
3.3.4.1 Exceptions	9
3.3.4.2 Syntactic containment	9
3.3.4.3 Terms denoting rule requirements	10
3.3.4.4 Rule evaluation order	10
3.3.4.5 Conditional rules	11
3.3.4.6 Syntactic substitution	11
3.3.4.7 Other terms	12
3.3.5 Descriptors	12
3.3.6 Index typography	13
3.4 Object identifier for Database Language SQL	13
4 Concepts	15
4.1 Data types	15
4.2 Character strings	16
4.2.1 Character strings and collating sequences	16

V

	<i>Page</i>
4.2.2 Operations involving character strings	17
4.2.2.1 Operators that operate on character strings and return character strings	17
4.2.2.2 Other operators involving character strings	18
4.2.3 Rules determining collating sequence usage	18
4.3 Bit strings	21
4.3.1 Bit string comparison and assignment	21
4.3.2 Operations involving bit strings	21
4.3.2.1 Operators that operate on bit strings and return bit strings	21
4.3.2.2 Other operators involving bit strings	21
4.4 Numbers	21
4.4.1 Characteristics of numbers	22
4.4.2 Operations involving numbers	23
4.5 Datetimes and intervals	23
4.5.1 Datetimes	23
4.5.2 Intervals	25
4.5.3 Operations involving datetimes and intervals	26
4.6 Type conversions and mixing of data types	27
4.7 Domains	28
4.8 Columns	28
4.9 Tables	29
4.10 Integrity constraints	31
4.10.1 Checking of constraints	32
4.10.2 Table constraints	32
4.10.3 Domain constraints	33
4.10.4 Assertions	33
4.11 SQL-schemas	34
4.12 Catalogs	34
4.13 Clusters of catalogs	35
4.14 SQL-data	35
4.15 SQL-environment	35
4.16 Modules	36
4.17 Procedures	36
4.18 Parameters	36
4.18.1 Status parameters	36
4.18.2 Data parameters	37
4.18.3 Indicator parameters	37
4.19 Diagnostics area	37
4.20 Standard programming languages	38
4.21 Cursors	38
4.22 SQL-statements	39
4.22.1 Classes of SQL-statements	39
4.22.2 SQL-statements classified by function	40
4.22.3 Embeddable SQL-statements	43
4.22.4 Preparable and immediately executable SQL-statements	44
4.22.5 Directly executable SQL-statements	46
4.22.6 SQL-statements and transaction states	47
4.23 Embedded syntax	48
4.24 SQL dynamic statements	49
4.25 Direct invocation of SQL	51

	<i>Page</i>
4.26 Privileges	51
4.27 SQL-agents	53
4.28 SQL-transactions	53
4.29 SQL-connections	56
4.30 SQL-sessions	57
4.31 Client-server operation	59
4.32 Information Schema	59
4.33 Leveling	60
4.34 SQL Flagger	60
5 Lexical elements	63
5.1 <SQL terminal character>	63
5.2 <token> and <separator>	66
5.3 <literal>	71
5.4 Names and identifiers	78
6 Scalar expressions	85
6.1 <data type>	85
6.2 <value specification> and <target specification>	91
6.3 <table reference>	94
6.4 <column reference>	96
6.5 <set function specification>	98
6.6 <numeric value function>	101
6.7 <string value function>	105
6.8 <datetime value function>	110
6.9 <case expression>	112
6.10 <cast specification>	114
6.11 <value expression>	124
6.12 <numeric value expression>	126
6.13 <string value expression>	128
6.14 <datetime value expression>	132
6.15 <interval value expression>	135
7 Query expressions	139
7.1 <row value constructor>	139
7.2 <table value constructor>	141
7.3 <table expression>	142
7.4 <from clause>	143
7.5 <joined table>	145
7.6 <where clause>	150
7.7 <group by clause>	151
7.8 <having clause>	153
7.9 <query specification>	155
7.10 <query expression>	159
7.11 <scalar subquery>, <row subquery>, and <table subquery>	165
8 Predicates	167
8.1 <predicate>	167
8.2 <comparison predicate>	169
8.3 <between predicate>	172

	<i>Page</i>
8.4 <in predicate>	173
8.5 <like predicate>	175
8.6 <null predicate>	178
8.7 <quantified comparison predicate>	180
8.8 <exists predicate>	182
8.9 <unique predicate>	183
8.10 <match predicate>	184
8.11 <overlaps predicate>	186
8.12 <search condition>	188
9 Data assignment rules	191
9.1 Retrieval assignment	191
9.2 Store assignment	193
9.3 Set operation result data types	195
10 Additional common elements	197
10.1 <interval qualifier>	197
10.2 <language clause>	201
10.3 <privileges>	203
10.4 <character set specification>	205
10.5 <collate clause>	207
10.6 <constraint name definition> and <constraint attributes>	208
11 Schema definition and manipulation	211
11.1 <schema definition>	211
11.2 <drop schema statement>	214
11.3 <table definition>	216
11.4 <column definition>	218
11.5 <default clause>	221
11.6 <table constraint definition>	224
11.7 <unique constraint definition>	226
11.8 <referential constraint definition>	228
11.9 <check constraint definition>	233
11.10 <alter table statement>	235
11.11 <add column definition>	236
11.12 <alter column definition>	237
11.13 <set column default clause>	238
11.14 <drop column default clause>	239
11.15 <drop column definition>	240
11.16 <add table constraint definition>	242
11.17 <drop table constraint definition>	243
11.18 <drop table statement>	244
11.19 <view definition>	245
11.20 <drop view statement>	249
11.21 <domain definition>	250
11.22 <alter domain statement>	252
11.23 <set domain default clause>	253
11.24 <drop domain default clause>	254
11.25 <add domain constraint definition>	255
11.26 <drop domain constraint definition>	256

	<i>Page</i>
11.27 <drop domain statement>	257
11.28 <character set definition>	259
11.29 <drop character set statement>	261
11.30 <collation definition>	262
11.31 <drop collation statement>	265
11.32 <translation definition>	267
11.33 <drop translation statement>	269
11.34 <assertion definition>	270
11.35 <drop assertion statement>	272
11.36 <grant statement>	273
11.37 <revoke statement>	276
12 Module	281
12.1 <module>	281
12.2 <module name clause>	284
12.3 <procedure>	285
12.4 Calls to a <procedure>	290
12.5 <SQL procedure statement>	303
13 Data manipulation	307
13.1 <declare cursor>	307
13.2 <open statement>	310
13.3 <fetch statement>	312
13.4 <close statement>	315
13.5 <select statement: single row>	316
13.6 <delete statement: positioned>	318
13.7 <delete statement: searched>	320
13.8 <insert statement>	322
13.9 <update statement: positioned>	325
13.10 <update statement: searched>	328
13.11 <temporary table declaration>	330
14 Transaction management	333
14.1 <set transaction statement>	333
14.2 <set constraints mode statement>	335
14.3 <commit statement>	337
14.4 <rollback statement>	339
15 Connection management	341
15.1 <connect statement>	341
15.2 <set connection statement>	344
15.3 <disconnect statement>	346
16 Session management	349
16.1 <set catalog statement>	349
16.2 <set schema statement>	350
16.3 <set names statement>	351
16.4 <set session authorization identifier statement>	352
16.5 <set local time zone statement>	354

	<i>Page</i>
17 Dynamic SQL	355
17.1 Description of SQL item descriptor areas	355
17.2 <allocate descriptor statement>	360
17.3 <deallocate descriptor statement>	362
17.4 <get descriptor statement>	363
17.5 <set descriptor statement>	366
17.6 <prepare statement>	369
17.7 <deallocate prepared statement>	375
17.8 <describe statement>	376
17.9 <using clause>	377
17.10 <execute statement>	383
17.11 <execute immediate statement>	385
17.12 <dynamic declare cursor>	387
17.13 <allocate cursor statement>	388
17.14 <dynamic open statement>	390
17.15 <dynamic fetch statement>	392
17.16 <dynamic close statement>	394
17.17 <dynamic delete statement: positioned>	395
17.18 <dynamic update statement: positioned>	396
17.19 <preparable dynamic delete statement: positioned>	398
17.20 <preparable dynamic update statement: positioned>	399
18 Diagnostics management	401
18.1 <get diagnostics statement>	401
19 Embedded SQL	411
19.1 <embedded SQL host program>	411
19.2 <embedded exception declaration>	418
19.3 <embedded SQL Ada program>	421
19.4 <embedded SQL C program>	424
19.5 <embedded SQL COBOL program>	428
19.6 <embedded SQL Fortran program>	431
19.7 <embedded SQL MUMPS program>	434
19.8 <embedded SQL Pascal program>	436
19.9 <embedded SQL PL/I program>	439
20 Direct invocation of SQL	443
20.1 <direct SQL statement>	443
20.2 <direct select statement: multiple rows>	447
21 Information Schema and Definition Schema	449
21.1 Introduction	449
21.2 Information Schema	450
21.2.1 INFORMATION_SCHEMA Schema	450
21.2.2 INFORMATION_SCHEMA_CATALOG_NAME base table	451
21.2.3 INFORMATION_SCHEMA_CATALOG_NAME_CARDINALITY assertion	452
21.2.4 SCHEMATA view	453
21.2.5 DOMAINS view	454
21.2.6 DOMAIN_CONSTRAINTS view	455
21.2.7 TABLES view	456

X

	<i>Page</i>
21.2.8 VIEWS view	457
21.2.9 COLUMNS view	458
21.2.10 TABLE_PRIVILEGES view	460
21.2.11 COLUMN_PRIVILEGES view	461
21.2.12 USAGE_PRIVILEGES view	462
21.2.13 TABLE_CONSTRAINTS view	463
21.2.14 REFERENTIAL_CONSTRAINTS view	464
21.2.15 CHECK_CONSTRAINTS view	465
21.2.16 KEY_COLUMN_USAGE view	466
21.2.17 ASSERTIONS view	467
21.2.18 CHARACTER_SETS view	468
21.2.19 COLLATIONS view	469
21.2.20 TRANSLATIONS view	470
21.2.21 VIEW_TABLE_USAGE view	471
21.2.22 VIEW_COLUMN_USAGE view	472
21.2.23 CONSTRAINT_TABLE_USAGE view	473
21.2.24 CONSTRAINT_COLUMN_USAGE view	474
21.2.25 COLUMN_DOMAIN_USAGE view	475
21.2.26 SQL_LANGUAGES view	476
21.2.27 SQL_IDENTIFIER domain	477
21.2.28 CHARACTER_DATA domain	477
21.2.29 CARDINAL_NUMBER domain	478
21.3 Definition Schema	479
21.3.1 Introduction	479
21.3.2 DEFINITION_SCHEMA Schema	480
21.3.3 USERS base table	481
21.3.4 SCHEMATA base table	482
21.3.5 DATA_TYPE_DESCRIPTOR base table	483
21.3.6 DOMAINS base table	485
21.3.7 DOMAIN_CONSTRAINTS base table	486
21.3.8 TABLES base table	488
21.3.9 VIEWS base table	489
21.3.10 COLUMNS base table	491
21.3.11 VIEW_TABLE_USAGE base table	493
21.3.12 VIEW_COLUMN_USAGE base table	494
21.3.13 TABLE_CONSTRAINTS base table	495
21.3.14 KEY_COLUMN_USAGE base table	497
21.3.15 REFERENTIAL_CONSTRAINTS base table	499
21.3.16 CHECK_CONSTRAINTS base table	501
21.3.17 CHECK_TABLE_USAGE base table	502
21.3.18 CHECK_COLUMN_USAGE base table	503
21.3.19 ASSERTIONS base table	504
21.3.20 TABLE_PRIVILEGES base table	505
21.3.21 COLUMN_PRIVILEGES base table	507
21.3.22 USAGE_PRIVILEGES base table	509
21.3.23 CHARACTER_SETS base table	511
21.3.24 COLLATIONS base table	513
21.3.25 TRANSLATIONS base table	515
21.3.26 SQL_LANGUAGES base table	517

	<i>Page</i>
21.4 Assertions on the base tables	520
21.4.1 UNIQUE_CONSTRAINT_NAME assertion	520
21.4.2 EQUAL_KEY_DEGREES assertion	521
21.4.3 KEY_DEGREE_GREATER_THAN_OR_EQUAL_TO_1 assertion	522
22 Status codes	523
22.1 SQLSTATE	523
22.2 SQLCODE	527
23 Conformance	529
23.1 Introduction	529
23.2 Claims of conformance	529
23.3 Extensions and options	530
23.4 Flagger requirements	530
23.5 Processing methods	530
Annex A Leveling the SQL Language	533
A.1 Intermediate SQL Specifications	533
A.2 Entry SQL Specifications	542
Annex B Implementation-defined elements	553
Annex C Implementation-dependent elements	565
Annex D Deprecated features	571
Annex E Incompatibilities with ISO/IEC 9075:1989	573
Annex F Maintenance and interpretation of SQL	579
Index	581

xii

TABLES

Table		<i>Page</i>
1	Collating coercibility rules for monadic operators	19
2	Collating coercibility rules for dyadic operators	19
3	Collating sequence usage for comparisons	20
4	Fields in datetime items	24
5	Fields in year-month INTERVAL items	25
6	Fields in day-time INTERVAL items	25
7	Valid values for fields in INTERVAL items	26
8	Valid operators involving datetimes and intervals	26
9	SQL-transaction isolation levels and the three phenomena	55
10	Valid values for fields in datetime items	89
11	Valid values for fields in INTERVAL items	90
12	<null predicate> semantics	178
13	Truth table for the AND boolean	189
14	Truth table for the OR boolean	189
15	Truth table for the IS boolean	189
16	Standard programming languages	201
17	Data types of <key word>s used in SQL item descriptor areas	356
18	Codes used for SQL data types in Dynamic SQL	358
19	Codes associated with datetime data types in Dynamic SQL	358
20	Codes used for <interval qualifier>s in Dynamic SQL	359
21	<identifier>s for use with <get diagnostics statement>	403
22	SQL-statement character codes for use in the diagnostics area	404
23	SQLSTATE class and subclass values	523
24	SQLCODE values	527

Information technology—Database languages—SQL

1 Scope

This 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 International Standard is described by the Reference Model of Data Management (ISO/IEC DIS 10032:1991).

This 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 International Standard

- provides a vehicle for portability of data definitions and compilation units between SQL-implementations,
- provides a vehicle for interconnection of SQL-implementations,
- 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, and
- specifies syntax for direct invocation of SQL-statements.

This 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.

2

Implementations of this 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 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 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.

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 11756:¹, *Information technology—Programming languages—MUMPS*.

¹ To be published.

3 Definitions, notations, and conventions

3.1 Definitions

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

3.1.1 Definitions taken from ISO/IEC 10646

This 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 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 International Standard

This International Standard defines the following terms:

- a) **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.
- b) **cardinality** (of a collection): The number of objects in that collection. Those objects need not necessarily have distinct values.
- c) **character repertoire; repertoire**: A set of characters used for a specific purpose or application. Each character repertoire has an implied default collating sequence.
- d) **coercibility**: An attribute of character string data items that governs how a collating sequence for the items is determined.
- e) **collation; collating sequence**: A method of ordering two comparable character strings. Every character set has a default collation.
- f) **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.

3.1 Definitions

- g) **descriptor**: A coded description of an SQL object. It includes all of the information about the object that a conforming SQL-implementation requires.
- h) **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.
- i) **duplicate**: Two or more values or rows are said to be duplicates (of each other) if and only if they are not distinct.
- j) **dyadic operator**: An operator having two operands: a left operand and a right operand. An example of a dyadic arithmetic operator in this International Standard is "-", specifying the subtraction of the right operand from the left operand.
- k) **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.
- l) **form-of-use conversion**: A method of converting character strings from one form-of-use to another form-of-use.
- m) **implementation-defined**: Possibly differing between SQL-implementations, but specified by the implementor for each particular SQL-implementation.
- n) **implementation-dependent**: Possibly differing between SQL-implementations, but not specified by this International Standard and not required to be specified by the implementor for any particular SQL-implementations.
- o) **monadic operator**: An operator having one operand. An example of a monadic arithmetic operator in this International Standard is "-", specifying the negation of the operand.
- p) **multiset**: An unordered collection of objects that are not necessarily distinct. The collection may be empty.
- q) ***n*-adic operator**: An operator having a variable number of operands (informally: *n* operands). An example of an *n*-adic operator in this International Standard is COALESCE.
- r) **null value (null)**: A special value, or mark, that is used to indicate the absence of any data value.
- s) **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 4.28, "SQL-transactions") or an SQL-session (see Subclause 4.30, "SQL-sessions") are not regarded as deliberate.
- t) **redundant duplicates**: All except one of any multiset of duplicate values or rows.
- u) **repertoire**: See **character repertoire**.
- v) **sequence**: An ordered collection of objects that are not necessarily distinct.
- w) **set**: An unordered collection of distinct objects. The collection may be empty.
- x) **SQL-implementation**: A database management system that conforms to this International Standard.

3.1 Definitions

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

3.2 Notation

The syntactic notation used in this International Standard is an extended version of BNF ("Backus Naur Form" or "Backus Normal 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 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 shall 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 ordinary 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.

3.2 Notation

- 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 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 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 shall hold before the General Rules may be successfully applied.
- **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 shall 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 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 12.3, “<procedure>”, and Subclause 20.1, “<direct SQL statement>”, may be terminated, diagnostic information is to be made available, and 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 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 if 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 that contains <C>.

In SQL language, an instance *A1* of <A> is said to *immediately contain* an instance *B1* of if <A> immediately contains 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 that contains *C1*.

An instance *A1* of <A> is said to contain an instance *B1* of *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 *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 if *A1* contains *B1* without an intervening instance *A2* of <A> or an intervening instance *B2* of .

3.3 Conventions

If $\langle A \rangle$ contains $\langle B \rangle$, then $\langle B \rangle$ is said to be *contained in* $\langle A \rangle$ and $\langle A \rangle$ is said to be a *containing* production symbol for $\langle B \rangle$. If $\langle A \rangle$ simply contains $\langle B \rangle$, then $\langle B \rangle$ is said to be *simply contained in* $\langle A \rangle$ and $\langle A \rangle$ is said to be a *simply containing* production symbol for $\langle B \rangle$.

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

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

If $\langle A \rangle$ contains a $\langle \text{table name} \rangle$ that identifies a view that is defined by a $\langle \text{view definition} \rangle V$, then $\langle A \rangle$ is said to *generally contain* the $\langle \text{query expression} \rangle$ contained in V . If $\langle A \rangle$ contains $\langle B \rangle$, then $\langle A \rangle$ generally contains $\langle B \rangle$. If $\langle A \rangle$ generally contains $\langle B \rangle$ and $\langle B \rangle$ generally contains $\langle C \rangle$, then $\langle A \rangle$ generally contains $\langle C \rangle$.

An instance $A1$ of $\langle A \rangle$ *directly contains* an instance $B1$ of $\langle B \rangle$ if $A1$ contains $B1$ without an intervening $\langle \text{set function specification} \rangle$ or $\langle \text{subquery} \rangle$.

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.

3.3 Conventions

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 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:

$\langle A \rangle ::= \langle B \rangle \langle C \rangle$

the Syntax Rules and Access Rules for $\langle A \rangle$, $\langle B \rangle$, and $\langle C \rangle$ are effectively applied simultaneously. The General Rules for $\langle B \rangle$ are applied before the General Rules for $\langle C \rangle$, and the General Rules for $\langle A \rangle$ are applied after the General Rules for both $\langle B \rangle$ and $\langle C \rangle$.

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 $\langle X \rangle$ is implicit: Y” indicates that the Syntax and General Rules are to be interpreted as if a syntactic element $\langle X \rangle$ 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

3.3 Conventions

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 <procedure> that has been executed, or if *S1* is the value of an <SQL statement variable> referenced by an <execute immediate statement> contained in a <procedure> 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 <procedure> that has been 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>, etc. 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.

3.3 Conventions

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 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>

```

3.4 Object identifier for Database Language SQL

<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.

A value is primitive in that it has no logical subdivision within this International Standard. A value is 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, FLOAT, REAL, DOUBLE PRECISION, DATE, TIME, TIMESTAMP, and INTERVAL.

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

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.

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 12.3, "<procedure>", and Subclause 19.1, "<embedded SQL host program>". Not every SQL data type has a corresponding data type in every host language.

4.2 Character strings

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 mutually comparable, subject to the restrictions specified in Table 3, “Collating sequence usage for comparisons”.

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”.

4.2 Character strings

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
- the name of the default collation of the character set.

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 set on which the collation operates,
- whether the collation has the NO PAD or the PAD SPACE attribute, and
- an indication of how the collation is performed.

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 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>.

4.2 Character strings

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 10.4, “`<character set specification>`”, and Subclause 11.28, “`<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.

4.2 Character strings

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

4.2 Character strings

Table 2—Collating coercibility rules for dyadic operators (Cont.)

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
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: <i>invalid syntax</i>	

Table 3—Collating sequence usage for comparisons

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	Coercible	default	default
Coercible	default	Implicit	Y	Y
Coercible	default	No collating sequence		Not permitted: <i>invalid syntax</i>
Coercible	default	Explicit	Y	Y
Implicit	X	Coercible	default	X
Implicit	X	Implicit	X	X
Implicit	X	Implicit	$Y \neq X$	Not permitted: <i>invalid syntax</i>
Implicit	X	No collating sequence		Not permitted: <i>invalid syntax</i>
No collating sequence		Explicit	Y	Y
No collating sequence		Any except Explicit	Any	Not permitted: <i>invalid syntax</i>
Explicit	X	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: <i>invalid syntax</i>

For n -adic operations (e.g., <case expression>) with operands $X_1, X_2, \dots, X_n$, 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);

4.4 Numbers

- 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^{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 6.12, "<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 4.5.3, "Operations involving datetimes and intervals") operates on a datetime or interval argument and returns an integer.

4.5 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.

4.5.1 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.

4.5 Datetimes and intervals

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 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

4.5 Datetimes and intervals

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.

4.5.2 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.

4.5 Datetimes and intervals

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
HOURL	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.

4.5.3 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

4.5 Datetimes and intervals

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.

4.6 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 6.12, "<numeric value expression>".

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 shall 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.

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.

4.6 Type conversions and mixing of data 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.

4.7 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 value of <default option>, if any, of the domain; and
- the domain constraint descriptors of the domain constraints, if any, of the domain.

4.8 Columns

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.

Every column has a <column name>.

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

A column has a nullability characteristic that indicates whether any attempt to store a null value into that column will inevitably raise an exception, or whether any attempt to retrieve a value from that column can ever result in a null value. A column *C* with <column name> *CN* of a base table *T* has a nullability characteristic that 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 expression> 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.

4.8 Columns

- *CN* is contained in a non-deferrable <unique constraint definition> whose <unique specification> specifies **PRIMARY KEY**.

Otherwise, a column *C* is *possibly nullable*.

A column is described by a column descriptor. A column 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 the column;
- the <collation name> from the <collate clause>, if any, of the column;
- the value of <default option>, if any, of the column;
- the nullability characteristic of the column; and
- the ordinal position of the column within the table that contains the column.

4.9 Tables

A table is a multiset of rows. A row is a nonempty sequence of values. 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.

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 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 tables* of derived tables and cursors are defined in Subclause 7.9, "<query specification>", Subclause 7.10, "<query expression>", and Subclause 13.1, "<declare cursor>". A viewed table has no *simply underlying tables*.

The *underlying tables* 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.

4.9 Tables

The *leaf underlying tables* 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 tables* 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 tables* 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.

A declared local temporary table is a named table defined by a <temporary table declaration> that is effectively materialized the first time any <procedure> in the <module> that contains the <temporary table declaration> is executed. A declared local temporary table is accessible only by <procedure>s in the <module> that contains the <temporary table declaration>. The effective <schema name> of the <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 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:

- 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; and
- the descriptor of each table constraint specified for the table.

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 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.

4.10 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.

4.10 Integrity constraints

4.10.1 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 *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>`.

4.10.2 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
- 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>`.

4.10 Integrity constraints

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 the 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.

4.10.3 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.

4.10.4 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.

4.10 Integrity constraints

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.

4.11 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 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, or a translation 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 <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.

4.12 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

4.12 Catalogs

<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.

4.13 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 definition 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 definition> and no <joined table> can “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.

4.14 SQL-data

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

4.15 SQL-environment

An *SQL-environment* comprises the following:

- an SQL-implementation capable of processing some Level (Entry SQL, Intermediate SQL, or Full SQL) of this International Standard and at least one binding style; see Clause 23, “Conformance” for further information about binding styles;
- 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.

4.16 Modules

4.16 Modules

A `<module>` is an object specified in the module language. A `<module>` is either a persistent `<module>` or an SQL-session `<module>`. The mechanisms by which `<module>`s are created or destroyed are implementation-defined. A `<module>` consists of an optional `<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 `<procedure>`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.

4.17 Procedures

A `<procedure>` consists of a `<procedure name>`, a sequence of `<parameter declaration>`s, and a single `<SQL procedure statement>`.

A `<procedure>` in a `<module>` is invoked by a compilation unit associated with the `<module>` by means of a host language “call” statement that specifies the `<procedure name>` of the `<procedure>` and supplies a sequence of parameter values corresponding in number and in `<data type>` to the `<parameter declaration>`s of the `<procedure>`. A call of a `<procedure>` causes the `<SQL procedure statement>` that it contains to be executed.

4.18 Parameters

A parameter is declared in a `<procedure>` 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 `<procedure>`. These `<data type>`s map to host language types and are not nullable except through the use of additional indicator variables.

4.18.1 Status parameters

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

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 International Standard. See Annex D, “Deprecated features”.