
**Information technology — Database
languages — SQL —**

**Part 2:
Foundation (SQL/Foundation)**

*Technologies de l'information — Langages de base de données — SQL —
Partie 2: Fondations (SQL/Fondations)*

Reference number
ISO/IEC 9075-2:1999(E)



PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 1999

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 734 10 79
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

Contents

Page

Foreword	xv
Introduction	xvii
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 Unicode	5
3.1.3 Definitions taken from ISO 8601	6
3.1.4 Definitions taken from Part 1	6
3.1.5 Definitions provided in Part 2	6
3.2 Notation	10
3.3 Conventions	10
3.3.1 Use of terms	10
3.3.1.1 Syntactic containment	10
4 Concepts	11
4.1 Data types	11
4.2 Character strings	13
4.2.1 Character strings and collating sequences	13
4.2.2 Operations involving character strings	14
4.2.2.1 Operators that operate on character strings and return character strings	14
4.2.2.2 Other operators involving character strings	15
4.2.2.3 Operations involving large object character strings	15
4.2.3 Rules determining collating sequence usage	16
4.2.4 Named character sets	18
4.3 Binary strings	20
4.3.1 Binary string comparison	21
4.3.2 Operations involving binary strings	21
4.3.2.1 Operators that operate on binary strings and return binary strings	21
4.3.2.2 Other operators involving binary strings	21
4.4 Bit strings	21
4.4.1 Bit string comparison and assignment	22

4.4.2	Operations involving bit strings	22
4.4.2.1	Operators that operate on bit strings and return bit strings	22
4.4.2.2	Other operators involving bit strings	22
4.5	Numbers	22
4.5.1	Characteristics of numbers	22
4.5.2	Operations involving numbers	23
4.6	Boolean types	24
4.6.1	Comparison and assignment of booleans	24
4.6.2	Operations involving booleans	24
4.6.2.1	Operations on booleans that return booleans	24
4.6.2.2	Other operators involving booleans	24
4.7	Datetimes and intervals	24
4.7.1	Datetimes	25
4.7.2	Intervals	27
4.7.3	Operations involving datetimes and intervals	29
4.8	User-defined types	30
4.8.1	Observers and mutators	32
4.8.2	Constructors	32
4.8.3	Subtypes and supertypes	32
4.8.4	User-defined type comparison and assignment	33
4.8.5	Transforms for user-defined types	34
4.9	Row types	35
4.10	Reference types	35
4.10.1	Operations involving references	36
4.11	Collection types	36
4.11.1	Arrays	36
4.11.2	Collection comparison	37
4.11.3	Operations involving collections	37
4.11.3.1	Operators that operate on array values and return array elements	37
4.11.3.2	Operators that operate on array values and return array values	37
4.12	Type conversions and mixing of data types	37
4.13	Data conversions	39
4.14	Domains	40
4.15	Columns, fields, and attributes	40
4.16	Tables	42
4.16.1	Types of tables	44
4.16.2	Referenceable tables, subtables, and supertables	45
4.16.3	Operations involving tables	46
4.17	Integrity constraints	48
4.17.1	Checking of constraints	48
4.17.2	Table constraints	49
4.17.3	Domain constraints	50
4.17.4	Assertions	50
4.18	Functional dependencies	50
4.18.1	General rules and definitions	51

4.18.2	Known functional dependencies in a base table	52
4.18.3	Known functional dependencies in <table value constructor>	53
4.18.4	Known functional dependencies in a <joined table>	53
4.18.5	Known functional dependencies in a <table reference>	54
4.18.6	Known functional dependencies in the result of a <from clause>	55
4.18.7	Known functional dependencies in the result of a <where clause>	55
4.18.8	Known functional dependencies in the result of a <group by clause>	56
4.18.9	Known functional dependencies in the result of a <having clause>	56
4.18.10	Known functional dependencies in a <query specification>	56
4.18.11	Known functional dependencies in a <query expression>	57
4.19	Candidate keys	58
4.20	SQL-schemas	59
4.21	SQL-client modules	59
4.22	Externally-invoked procedures	60
4.23	SQL-invoked routines	61
4.24	Built-in functions	67
4.25	SQL-paths	67
4.26	Host parameters	68
4.26.1	Status parameters	68
4.26.2	Data parameters	69
4.26.3	Indicator parameters	69
4.26.4	Locators	69
4.27	Diagnostics area	70
4.28	Standard programming languages	70
4.29	Cursors	71
4.30	SQL-statements	73
4.30.1	Classes of SQL-statements	73
4.30.2	SQL-statements classified by function	73
4.30.3	SQL-statements and transaction states	76
4.30.4	SQL-statement atomicity	77
4.31	Basic security model	77
4.31.1	Authorization identifiers	77
4.31.1.1	SQL-session authorization identifiers	78
4.31.1.2	SQL-client module authorization identifiers	79
4.31.1.3	SQL-schema authorization identifiers	79
4.31.2	Privileges	79
4.31.3	Roles	81
4.31.4	Security model definitions	81
4.32	SQL-transactions	82
4.33	SQL-connections	86
4.34	SQL-sessions	87
4.34.1	Execution contexts	89
4.35	Triggers	90
4.35.1	Triggered actions	90
4.35.2	Execution of triggers	91

4.36	Client-server operation	92
5	Lexical elements	93
5.1	<SQL terminal character>	93
5.2	<token> and <separator>	96
5.3	<literal>	105
5.4	Names and identifiers	113
6	Scalar expressions	121
6.1	<data type>	121
6.2	<field definition>	130
6.3	<value specification> and <target specification>	132
6.4	<contextually typed value specification>	136
6.5	<identifier chain>	138
6.6	<column reference>	141
6.7	<SQL parameter reference>	143
6.8	<field reference>	144
6.9	<attribute or method reference>	145
6.10	<method reference>	146
6.11	<method invocation>	147
6.12	<static method invocation>	149
6.13	<element reference>	151
6.14	<dereference operation>	152
6.15	<reference resolution>	153
6.16	<set function specification>	155
6.17	<numeric value function>	159
6.18	<string value function>	164
6.19	<datetime value function>	175
6.20	<interval value function>	177
6.21	<case expression>	178
6.22	<cast specification>	181
6.23	<value expression>	197
6.24	<new specification>	200
6.25	<subtype treatment>	201
6.26	<numeric value expression>	202
6.27	<string value expression>	204
6.28	<datetime value expression>	209
6.29	<interval value expression>	212
6.30	<boolean value expression>	216
6.31	<array value expression>	219
6.32	<array value constructor>	221
7	Query expressions	223
7.1	<row value constructor>	223
7.2	<row value expression>	226
7.3	<table value constructor>	227

7.4	<table expression>	229
7.5	<from clause>	230
7.6	<table reference>	232
7.7	<joined table>	238
7.8	<where clause>	244
7.9	<group by clause>	245
7.10	<having clause>	256
7.11	<query specification>	258
7.12	<query expression>	265
7.13	<search or cycle clause>	279
7.14	<subquery>	283
8	Predicates	285
8.1	<predicate>	285
8.2	<comparison predicate>	287
8.3	<between predicate>	295
8.4	<in predicate>	296
8.5	<like predicate>	298
8.6	<similar predicate>	304
8.7	<null predicate>	309
8.8	<quantified comparison predicate>	310
8.9	<exists predicate>	312
8.10	<unique predicate>	313
8.11	<match predicate>	314
8.12	<overlaps predicate>	316
8.13	<distinct predicate>	318
8.14	<type predicate>	320
8.15	<search condition>	322
9	Data assignment rules and routine determination	323
9.1	Retrieval assignment	323
9.2	Store assignment	328
9.3	Data types of results of aggregations	333
9.4	Subject routine determination	336
9.5	Type precedence list determination	337
9.6	Host parameter mode determination	342
9.7	Type name determination	344
10	Additional common elements	347
10.1	<interval qualifier>	347
10.2	<language clause>	351
10.3	<path specification>	353
10.4	<routine invocation>	354
10.5	<privileges>	374
10.6	<character set specification>	379
10.7	<specific routine designator>	381

10.8	<collate clause>	384
10.9	<constraint name definition> and <constraint characteristics>	385
10.10	Execution of BEFORE triggers	387
10.11	Execution of AFTER triggers	388
10.12	Execution of triggers	389
10.13	Execution of array-returning functions	390
10.14	Data type identity	393
10.15	Determination of a from-sql function	395
10.16	Determination of a from-sql function for an overriding method	396
10.17	Determination of a to-sql function	397
10.18	Determination of a to-sql function for an overriding method	398
11	Schema definition and manipulation	399
11.1	<schema definition>	399
11.2	<drop schema statement>	402
11.3	<table definition>	404
11.4	<column definition>	412
11.5	<default clause>	418
11.6	<table constraint definition>	422
11.7	<unique constraint definition>	424
11.8	<referential constraint definition>	426
11.9	<check constraint definition>	440
11.10	<alter table statement>	442
11.11	<add column definition>	444
11.12	<alter column definition>	445
11.13	<set column default clause>	446
11.14	<drop column default clause>	447
11.15	<add column scope clause>	448
11.16	<drop column scope clause>	449
11.17	<drop column definition>	451
11.18	<add table constraint definition>	453
11.19	<drop table constraint definition>	454
11.20	<drop table statement>	456
11.21	<view definition>	459
11.22	<drop view statement>	469
11.23	<domain definition>	471
11.24	<alter domain statement>	474
11.25	<set domain default clause>	475
11.26	<drop domain default clause>	476
11.27	<add domain constraint definition>	477
11.28	<drop domain constraint definition>	478
11.29	<drop domain statement>	479
11.30	<character set definition>	481
11.31	<drop character set statement>	483
11.32	<collation definition>	485
11.33	<drop collation statement>	487

11.34	<translation definition>	489
11.35	<drop translation statement>	491
11.36	<assertion definition>	493
11.37	<drop assertion statement>	495
11.38	<trigger definition>	497
11.39	<drop trigger statement>	501
11.40	<user-defined type definition>	502
11.41	<attribute definition>	517
11.42	<alter type statement>	520
11.43	<add attribute definition>	521
11.44	<drop attribute definition>	523
11.45	<add original method specification>	525
11.46	<add overriding method specification>	531
11.47	<drop method specification>	535
11.48	<drop data type statement>	537
11.49	<SQL-invoked routine>	541
11.50	<alter routine statement>	562
11.51	<drop routine statement>	565
11.52	<user-defined cast definition>	567
11.53	<drop user-defined cast statement>	569
11.54	<user-defined ordering definition>	571
11.55	<drop user-defined ordering statement>	574
11.56	<transform definition>	576
11.57	<drop transform statement>	579
12	Access control	583
12.1	<grant statement>	583
12.2	<grant privilege statement>	588
12.3	<role definition>	591
12.4	<grant role statement>	592
12.5	<drop role statement>	594
12.6	<revoke statement>	595
13	SQL-client modules	611
13.1	<SQL-client module definition>	611
13.2	<module name clause>	615
13.3	<externally-invoked procedure>	616
13.4	Calls to an <externally-invoked procedure>	619
13.5	<SQL procedure statement>	633
13.6	Data type correspondences	641
14	Data manipulation	651
14.1	<declare cursor>	651
14.2	<open statement>	657
14.3	<fetch statement>	659
14.4	<close statement>	663

14.5	<select statement: single row>	665
14.6	<delete statement: positioned>	667
14.7	<delete statement: searched>	670
14.8	<insert statement>	673
14.9	<update statement: positioned>	677
14.10	<update statement: searched>	684
14.11	<temporary table declaration>	690
14.12	<free locator statement>	692
14.13	<hold locator statement>	693
14.14	Effect of deleting rows from base tables	694
14.15	Effect of deleting some rows from a derived table	696
14.16	Effect of deleting some rows from a viewed table	698
14.17	Effect of inserting tables into base tables	699
14.18	Effect of inserting a table into a derived table	701
14.19	Effect of inserting a table into a viewed table	703
14.20	Effect of replacing rows in base tables	704
14.21	Effect of replacing some rows in a derived table	706
14.22	Effect of replacing some rows in a viewed table	708
15	Control statements	711
15.1	<call statement>	711
15.2	<return statement>	712
16	Transaction management	715
16.1	<start transaction statement>	715
16.2	<set transaction statement>	717
16.3	<set constraints mode statement>	719
16.4	<savepoint statement>	721
16.5	<release savepoint statement>	722
16.6	<commit statement>	723
16.7	<rollback statement>	725
17	Connection management	727
17.1	<connect statement>	727
17.2	<set connection statement>	730
17.3	<disconnect statement>	732
18	Session management	735
18.1	<set session characteristics statement>	735
18.2	<set session user identifier statement>	736
18.3	<set role statement>	737
18.4	<set local time zone statement>	738
19	Diagnostics management	739
19.1	<get diagnostics statement>	739

20	Information Schema	751
20.1	Introduction to Information Schema and Definition Schema	751
20.2	INFORMATION_SCHEMA Schema	752
20.3	INFORMATION_SCHEMA_CATALOG_NAME base table	753
20.4	CARDINAL_NUMBER domain	754
20.5	CHARACTER_DATA domain	754
20.6	SQL_IDENTIFIER domain	755
20.7	TIME_STAMP domain	755
20.8	ADMINISTRABLE_ROLE_AUTHORIZATIONS view	756
20.9	APPLICABLE_ROLES view	757
20.10	ASSERTIONS view	758
20.11	ATTRIBUTES view	759
20.12	CHARACTER_SETS view	761
20.13	CHECK_CONSTRAINTS view	762
20.14	COLLATIONS view	763
20.15	COLUMN_DOMAIN_USAGE view	764
20.16	COLUMN_PRIVILEGES view	765
20.17	COLUMN_UDT_USAGE view	766
20.18	COLUMNS view	767
20.19	CONSTRAINT_COLUMN_USAGE view	769
20.20	CONSTRAINT_TABLE_USAGE view	770
20.21	DATA_TYPE_PRIVILEGES view	771
20.22	DIRECT_SUPERTABLES view	772
20.23	DIRECT_SUPERTYPES view	773
20.24	DOMAIN_CONSTRAINTS view	774
20.25	DOMAIN_UDT_USAGE view	775
20.26	DOMAINS view	776
20.27	ELEMENT_TYPES view	777
20.28	ENABLED_ROLES view	778
20.29	FIELDS view	779
20.30	KEY_COLUMN_USAGE view	780
20.31	METHOD_SPECIFICATION_PARAMETERS view	781
20.32	METHOD_SPECIFICATIONS view	783
20.33	PARAMETERS view	785
20.34	REFERENCED_TYPES view	787
20.35	REFERENTIAL_CONSTRAINTS view	788
20.36	ROLE_COLUMN_GRANTS view	789
20.37	ROLE_ROUTINE_GRANTS view	790
20.38	ROLE_TABLE_GRANTS view	791
20.39	ROLE_TABLE_METHOD_GRANTS view	792
20.40	ROLE_USAGE_GRANTS view	793
20.41	ROLE_UDT_GRANTS view	794
20.42	ROUTINE_COLUMN_USAGE view	795
20.43	ROUTINE_PRIVILEGES view	796
20.44	ROUTINE_TABLE_USAGE view	797

20.45	ROUTINES view	798
20.46	SCHEMATA view	800
20.47	SQL_FEATURES view	801
20.48	SQL_IMPLEMENTATION_INFO view	802
20.49	SQL_LANGUAGES view	803
20.50	SQL_PACKAGES view	804
20.51	SQL_SIZING view	805
20.52	SQL_SIZING_PROFILES view	806
20.53	TABLE_CONSTRAINTS view	807
20.54	TABLE_METHOD_PRIVILEGES view	808
20.55	TABLE_PRIVILEGES view	809
20.56	TABLES view	810
20.57	TRANSFORMS view	811
20.58	TRANSLATIONS view	812
20.59	TRIGGERED_UPDATE_COLUMNS view	813
20.60	TRIGGER_COLUMN_USAGE view	814
20.61	TRIGGER_TABLE_USAGE view	815
20.62	TRIGGERS view	816
20.63	USAGE_PRIVILEGES view	817
20.64	UDT_PRIVILEGES view	818
20.65	USER_DEFINED_TYPES view	819
20.66	VIEW_COLUMN_USAGE view	821
20.67	VIEW_TABLE_USAGE view	822
20.68	VIEWS view	823
20.69	Short name views	824
20.70	Definition of SQL built-in functions	835
21	Definition Schema	847
21.1	Introduction to the Definition Schema	847
21.2	DEFINITION_SCHEMA Schema	848
21.3	EQUAL_KEY_DEGREES assertion	849
21.4	KEY_DEGREE_GREATER_THAN_OR_EQUAL_TO_1 assertion	850
21.5	UNIQUE_CONSTRAINT_NAME assertion	851
21.6	ASSERTIONS base table	852
21.7	ATTRIBUTES base table	853
21.8	CHARACTER_SETS base table	855
21.9	CHECK_COLUMN_USAGE base table	857
21.10	CHECK_TABLE_USAGE base table	858
21.11	CHECK_CONSTRAINTS base table	859
21.12	COLLATIONS base table	860
21.13	COLUMN_PRIVILEGES base table	862
21.14	COLUMNS base table	864
21.15	DATA_TYPE_DESCRIPTOR base table	867
21.16	DIRECT_SUPERTABLES base table	874
21.17	DIRECT_SUPERTYPES base table	876
21.18	DOMAIN_CONSTRAINTS base table	878

21.19	DOMAINS base table	880
21.20	ELEMENT_TYPES base table	881
21.21	FIELDS base table	882
21.22	KEY_COLUMN_USAGE base table	884
21.23	METHOD_SPECIFICATION_PARAMETERS base table	886
21.24	METHOD_SPECIFICATIONS base table	888
21.25	PARAMETERS base table	891
21.26	REFERENCED_TYPES base table	893
21.27	REFERENTIAL_CONSTRAINTS base table	894
21.28	ROLE_AUTHORIZATION_DESCRIPTOR base table	896
21.29	ROLES base table	898
21.30	ROUTINE_COLUMN_USAGE base table	899
21.31	ROUTINE_PRIVILEGES base table	901
21.32	ROUTINE_TABLE_USAGE base table	903
21.33	ROUTINES base table	905
21.34	SCHEMATA base table	910
21.35	SQL_FEATURES base table	911
21.36	SQL_IMPLEMENTATION_INFO base table	913
21.37	SQL_LANGUAGES base table	914
21.38	SQL_SIZING base table	918
21.39	SQL_SIZING_PROFILES base table	919
21.40	TABLE_CONSTRAINTS base table	920
21.41	TABLE_METHOD_PRIVILEGES base table	922
21.42	TABLE_PRIVILEGES base table	924
21.43	TABLES base table	926
21.44	TRANSFORMS base table	928
21.45	TRANSLATIONS base table	929
21.46	TRIGGERED_UPDATE_COLUMNS base table	931
21.47	TRIGGER_COLUMN_USAGE base table	932
21.48	TRIGGER_TABLE_USAGE base table	934
21.49	TRIGGERS base table	936
21.50	USAGE_PRIVILEGES base table	938
21.51	USER_DEFINED_TYPE_PRIVILEGES base table	940
21.52	USER_DEFINED_TYPES base table	942
21.53	USERS base table	945
21.54	VIEW_COLUMN_USAGE base table	946
21.55	VIEW_TABLE_USAGE base table	947
21.56	VIEWS base table	948
22	Status codes	951
22.1	SQLSTATE	951
22.2	Remote Database Access SQLSTATE Subclasses	958
22.3	SQL Multimedia and Application Packages SQLSTATE Subclasses	958

23	Conformance	961
23.1	General conformance requirements	961
23.2	Claims of conformance	962
Annex A	SQL Conformance Summary	965
Annex B	Implementation-defined elements	1017
Annex C	Implementation-dependent elements	1027
Annex D	Deprecated features	1033
Annex E	Incompatibilities with ISO/IEC 9075:1992 and ISO/IEC 9075-4:1996	1035
Annex F	SQL feature and package taxonomy	1041
Index		1065

TABLES

Tables	Page
1 Collating coercibility rules for monadic operators	17
2 Collating coercibility rules for dyadic operators	17
3 Collating sequence usage for comparisons	18
4 Fields in datetime values	25
5 Datetime data type conversions	27
6 Fields in year-month INTERVAL values	27
7 Fields in day-time INTERVAL values	28
8 Valid values for fields in INTERVAL values	28
9 Valid operators involving datetimes and intervals	29
10 SQL-transaction isolation levels and the three phenomena	84
11 Valid values for datetime fields	127
12 Valid absolute values for interval fields	128
13 Truth table for the AND boolean operator	218
14 Truth table for the OR boolean operator	218
15 Truth table for the IS boolean operator	218
16 <null predicate> semantics	309
17 Standard programming languages	352
18 Data type correspondences for Ada	641
19 Data type correspondences for C	642
20 Data type correspondences for COBOL	644
21 Data type correspondences for Fortran	646
22 Data type correspondences for MUMPS	647
23 Data type correspondences for Pascal	648
24 Data type correspondences for PL/I	649
25 <identifier>s for use with <get diagnostics statement>	741
26 SQL-statement codes	742
27 SQLSTATE class and subclass values	952
28 SQLSTATE class codes for RDA	958
29 SQLSTATE class codes for SQL/MM	959
30 Implied feature relationships	961
31 SQL/Foundation feature taxonomy and definition for Core SQL	1042
32 SQL/Foundation feature taxonomy for features outside Core SQL	1057

Foreword

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

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

International Standard ISO/IEC 9075-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

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

ISO/IEC 9075 consists of the following parts, under the general title *Information technology — Database languages — SQL*:

- *Part 1: Framework (SQL/Framework)*
- *Part 2: Foundation (SQL/Foundation)*
- *Part 3: Call-Level Interface (SQL/CLI)*
- *Part 4: Persistent Stored Modules (SQL/PSM)*
- *Part 5: Host Language Bindings (SQL/Bindings)*

Annexes A, B, C, D, E, and F of this part of ISO/IEC 9075 are for information only.

Introduction

The organization of this part of ISO/IEC 9075 is as follows:

- 1) Clause 1, “Scope”, specifies the scope of this part of ISO/IEC 9075.
- 2) Clause 2, “Normative references”, identifies additional standards that, through reference in this part of ISO/IEC 9075, constitute provisions of this part of ISO/IEC 9075.
- 3) Clause 3, “Definitions, notations, and conventions”, defines the notations and conventions used in this part of ISO/IEC 9075.
- 4) Clause 4, “Concepts”, presents concepts used in the definition of SQL.
- 5) Clause 5, “Lexical elements”, defines the lexical elements of the language.
- 6) Clause 6, “Scalar expressions”, defines the elements of the language that produce scalar values.
- 7) Clause 7, “Query expressions”, defines the elements of the language that produce rows and tables of data.
- 8) Clause 8, “Predicates”, defines the predicates of the language.
- 9) Clause 9, “Data assignment rules and routine determination”, specifies the rules for assignments that retrieve data from or store data into SQL-data, and formation rules for set operations.
- 10) Clause 10, “Additional common elements”, defines additional language elements that are used in various parts of the language.
- 11) Clause 11, “Schema definition and manipulation”, defines facilities for creating and managing a schema.
- 12) Clause 12, “Access control”, defines facilities for controlling access to SQL-data.
- 13) Clause 13, “SQL-client modules”, defines SQL-client modules and externally-invoked procedures.
- 14) Clause 14, “Data manipulation”, defines the data manipulation statements.
- 15) Clause 15, “Control statements”, defines the SQL-control statements.
- 16) Clause 16, “Transaction management”, defines the SQL-transaction management statements.
- 17) Clause 17, “Connection management” defines the SQL-connection management statements.
- 18) Clause 18, “Session management”, defines the SQL-session management statements.
- 19) Clause 19, “Diagnostics management”, defines the diagnostics management facilities.
- 20) Clause 20, “Information Schema”, defines viewed tables that contain schema information.
- 21) Clause 21, “Definition Schema”, defines base tables on which the viewed tables containing schema information depend.

- 22) Clause 22, “Status codes”, defines values that identify the status of the execution of SQL-statements and the mechanisms by which those values are returned.
- 23) Clause 23, “Conformance”, defines the criteria for conformance to this part of ISO/IEC 9075.
- 24) Annex A, “SQL Conformance Summary”, is an informative Annex. It summarizes the conformance requirements of the SQL language.
- 25) Annex B, “Implementation-defined elements”, is an informative Annex. It lists those features for which the body of this part of ISO/IEC 9075 states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or any other behavior is partly or wholly implementation-defined.
- 26) Annex C, “Implementation-dependent elements”, is an informative Annex. It lists those features for which the body of this part of ISO/IEC 9075 states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or any other behavior is partly or wholly implementation-dependent.
- 27) Annex D, “Deprecated features”, is an informative Annex. It lists features that the responsible Technical Committee intend will not appear in a future revised version of this part of ISO/IEC 9075.
- 28) Annex E, “Incompatibilities with ISO/IEC 9075:1992 and ISO/IEC 9075-4:1996”, is an informative Annex. It lists incompatibilities with the previous version of this part of ISO/IEC 9075.
- 29) Annex F, “SQL feature and package taxonomy”, is an informative Annex. It identifies features and packages of the SQL language specified in this part of ISO/IEC 9075 by an identifier and a short descriptive name. This taxonomy is used to specify conformance both to Core SQL and to the packages specified in this part of ISO/IEC 9075. The feature taxonomy may be used to develop other profiles involving the SQL language.

In the text of this part of ISO/IEC 9075, Clauses begin a new odd-numbered page, and in Clause 5, “Lexical elements”, through Clause 22, “Status codes”, Subclauses begin a new page. Any resulting blank space is not significant.

Information technology — Database languages — SQL —

Part 2:

Foundation (SQL/Foundation)

1 Scope

This part of ISO/IEC 9075 defines the data structures and basic operations on SQL-data. It provides functional capabilities for creating, accessing, maintaining, controlling, and protecting SQL-data.

This part of ISO/IEC 9075 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.
- For declaring database language procedures.

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

This part of ISO/IEC 9075 provides a vehicle for portability of data definitions and compilation units between SQL-implementations.

This part of ISO/IEC 9075 provides a vehicle for interconnection of SQL-implementations.

This part of ISO/IEC 9075 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.

Implementations of this part of ISO/IEC 9075 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 part of ISO/IEC 9075. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 9075 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated 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 1539-1:1997, *Information technology — Programming languages — Fortran — Part 1: Base language*.

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

ISO/IEC 2022:1994, *Information technology — Character code structure and extension techniques*.

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

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

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

ISO/IEC 8649:1996, *Information technology — Open Systems Interconnection — Service Definition for the Association Control Service Element*.

ISO/IEC 8652:1995, *Information technology — Programming languages — Ada*.

ISO 8824-1:1995, *Information technology — Specification of Abstract Syntax Notation One (ASN.1) — Part 1: Specification of basic notation*

ISO/IEC 9075-1:1999, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*.

ISO/IEC FDIS 9075-3:1999, *Information technology — Database languages — SQL — Part 3: Call-Level Interface (SQL/CLI)*.

ISO/IEC 9075-4:1999, *Information technology — Database languages — SQL — Part 4: Persistent Stored Modules (SQL/PSM)*.

ISO/IEC 9075-5:1999, *Information technology — Database languages — SQL — Part 5: Host Language Bindings (SQL/Bindings)*.

ISO/IEC 9579-1:1993, *Information technology — Open Systems Interconnection — Remote Database Access — Part 1: Generic Model, Service, and Protocol.*

ISO/IEC 9579-2:1998, *Information technology — Open Systems Interconnection — Remote Database Access — Part 2: SQL Specialization.*

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

ISO/IEC 9899:1990/Amendment 1:1995, *Amendment to ISO/IEC 9899:1990 — C Integrity.*

ISO/IEC 10026-2:1996, *Information technology — Open Systems Interconnection — Distributed Transaction Processing — Part 2: OSI TP Service.*

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

ISO/IEC 11404:1996, *Information technology — Programming languages, their environments and system software interfaces — Language-independent datatypes.*

ISO/IEC 11756:1992, *Information technology — Programming languages — MUMPS.*

ISO/IEC 10646-1:1993, *Information technology — Universal Multi-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.*

The Unicode Consortium, *The Unicode Standard, Version 2.0*, 1996. ISBN 0-201-48345-9.

3 Definitions, notations, and conventions

3.1 Definitions

For the purposes of this part of ISO/IEC 9075, the following definitions apply.

3.1.1 Definitions taken from ISO/IEC 10646

This part of ISO/IEC 9075 makes use of the following terms defined in ISO/IEC 10646:

- a) **character**
- b) **coded character**
- c) **coded character set**
- d) **control function**
- e) **private use plane**
- f) **repertoire**

3.1.2 Definitions taken from Unicode

This part of ISO/IEC 9075 makes use of the following terms defined in The Unicode Standard:

- a) **abstract character**
- b) **abstract character sequence**
- c) **canonical decomposition**
- d) **canonical equivalent**
- e) **code value**
- f) **compatibility decomposition**
- g) **compatibility equivalent**
- h) **control character**
- i) **private use**
- j) **surrogate pair**

3.1 Definitions

3.1.3 Definitions taken from ISO 8601

This part of ISO/IEC 9075 makes use of the following terms defined in ISO 8601:

- a) **Coördinated Universal Time** (UTC)
- b) **date** (“date, calendar” in ISO 8601)

3.1.4 Definitions taken from Part 1

This part of ISO/IEC 9075 makes use of all terms defined in ISO/IEC 9075-1.

3.1.5 Definitions provided in Part 2

This part of ISO/IEC 9075 defines the following terms:

- a) **assignable**: The characteristic of a data type that permits a value of that data type to be assigned to a site of a specified data type. See Subclause 4.12, “Type conversions and mixing of data types”.
- b) **assignment**: An operation that replaces the instance at a site (known as the *target*) with a new instance of a (possibly, though not necessarily, different) value (known as the *source*).
- c) **attribute**: A component (of a structured type) whose characteristics are specified by an attribute descriptor. A value *V* in structured type *T* has exactly one attribute value for each attribute *A* of *T*, this being the result of the invocation of *A(V)* of the observer function for that attribute.
- d) **cardinality** (of a value of a collection type): The number of elements in that value. Those elements need not necessarily have distinct values.
- e) **character repertoire; repertoire**: A set of characters used for a specific purpose or application. Each character repertoire has an implied default collating sequence.
- f) **coercibility**: A characteristic of a character string value that governs how a collating sequence for the value is determined.
- g) **collation; collating sequence**: A method of ordering two comparable character strings. Every character set has a default collation.
- h) **collection type**: The type of collection denoted by a collection data type: array.
- i) **comparable**: The characteristic of values that permits one value to be compared with another value. Also said of data types: Two data types are comparable if values of those data types are comparable. If one of the two data types is a user-defined type, then both shall be in the same subtype family. See Subclause 4.12, “Type conversions and mixing of data types”.
- j) **constructor function**: A niladic SQL-invoked function of which exactly one is implicitly specified for every structured type. An invocation of the constructor function for data type *T* returns a value *V* of the most specific type *T* such that *V* is not null and, for every observer function *O* defined for *T*, the invocation *O(V)* returns the default value of the attribute corresponding to *O*.

- k) **declared type:** Of an expression denoting a value, the unique data type that is common to every value that might result from evaluation of that expression. The term is also applicable to anything that can be referenced to denote a value, such as, for example, a parameter, column or variable.
- l) **distinct:** Two values other than rows and collections 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. Two arrays are distinct if the arrays do not have the same cardinality or if, for arrays of the same cardinality, elements in the same ordinal position in the two arrays are distinct; otherwise, the arrays are not distinct. The result of evaluating whether or not two values, two rows, or two arrays are distinct is never unknown.
- m) **duplicate:** Two or more values or rows are said to be duplicates (of each other) if and only if they are not distinct.
- n) **dyadic:** Of operators, functions, and procedures, having exactly two operands or parameters. An example of a dyadic operator in this part of ISO/IEC 9075 is “–”, specifying the subtraction of the right operand from the left operand. An example of a dyadic function is POSITION.
- o) **element type:** The definition of a collection type *CT* includes specification of a data type that is common to every element of every value in *CT*. This is the element type of *CT* and also the element type of every value in *CT*.
- p) **external routine:** An SQL-invoked routine whose routine body is an external body reference that identifies a program written in a standard programming language other than SQL.
- q) **fixed-length:** A characteristic of bit strings and character strings that restricts a string to contain exactly one number of bits or characters, respectively, known as the length in bits or characters, respectively, of the string.
- r) **form-of-use:** A convention (or encoding) for representing characters (in character strings). Some forms-of-use are fixed-length codings and others are variable-length codings.
- s) **form-of-use conversion:** A method of converting character strings from one form-of-use to another form-of-use.
- t) **interface (of a structured type):** The set comprising every function such that the declared type of at least one of its parameters or result is that structured type.
- u) **monadic:** Of operators, functions, and procedures, having exactly one operand or parameter. An example of a monadic arithmetic operator in this part of ISO/IEC 9075 is “–”, specifying the negation of the operand. An example of a monadic function is CHARACTER_LENGTH, specifying the length in characters of the argument.
- v) **most specific type:** Of a value, the unique data type of which every data type of that value is a supertype.
- w) **mutator function:** A dyadic, type-preserving function *M* whose definition is implied by the definition of some attribute *A* (of declared type *AT*) of some user-defined type *T*. The first parameter of *M* is a result SQL parameter of declared type *T*, which is also the result type of *M*. The second parameter of *M* is of declared type *AT*. The invocation *M*(*V*, *AV*) returns the value *V1* such that *V1* differs from *V* only in its value for attribute *A*, if at all. The most specific type of *V1* is the most specific type of *V*.

3.1 Definitions

- x) ***n*-adic operator**: An operator having a variable number of operands (informally: *n* operands). An example of an *n*-adic operator in this part of ISO/IEC 9075 is COALESCE.
- y) **niladic**: Of functions and procedures, having no parameters.
- z) **observer function**: An SQL-invoked function *A* implicitly defined by the definition of attribute *A* of a structured type *T*. If *V* is some value in *T* and the declared type of *A* is *AT*, then the invocation of *A*(*V*) returns some value *AV* in *AT*. *AV* is then said to be the value of attribute *A* in *V*.
- aa) **redundant duplicates**: All except one of any multiset of duplicate values or rows.
- bb) **reference type**: A data type all of whose values are potential references to sites of one specified data type.
- cc) **REF value**: A value that references some site.
- dd) **referenced type**: The declared type of the values at sites referenced by values of a particular reference type.
- ee) **referenced value**: The value at the site referenced by a REF value.
- ff) **repertoire**: See **character repertoire**.
- gg) **result SQL parameter**: An SQL parameter that specifies RESULT.
- hh) **result data type**: The result data type of an SQL-invoked function is the declared type of the result of its invocation.
- ii) **signature (of an SQL-invoked routine)**: the name of an SQL-invoked routine, the position and declared type of each of its SQL parameters, and an indication of whether it is an SQL-invoked function or an SQL-invoked procedure.
- jj) **SQL argument** An expression denoting a value to be substituted for an SQL parameter in an invocation of an SQL-invoked routine.
- kk) **SQL-invoked routine**: A routine that is allowed to be invoked only from within SQL.
- ll) **SQL parameter** An parameter declared as part of the signature of an SQL-invoked routine.
- mm) **SQL routine**: An SQL-invoked routine whose routine body is written in SQL.
- nn) **STATE function**: A dyadic BOOLEAN function, with <schema qualified routine name> EQUALS, which can be implicitly defined in connection with a structured type whose values are unordered. The parameters of the STATE function for data type *T* are both of declared type *T*. The invocation EQUALS(*V1*, *V2*) returns *true* if and only if, for every attribute *A* of *T*, *A*(*V1*) = *A*(*V2*).
- oo) **subfield**: A field *F* is a subfield of a row type *RT* if *F* is a field of *RT* or *F* is a field of a row type *RT2* that is the declared type of a field *F2* that is a subfield of *RT*.
- pp) **subtype**: A data type *T2* is a subtype of data type *T1* if every value of *T2* is also a value of *T1*. If *T1* and *T2* are not compatible, then *T2* is a *proper subtype* of *T1*. “Compatible” is defined in Subclause 4.12, “Type conversions and mixing of data types”. See also **supertype**.

- qq) **supertype**: A data type *T1* is a supertype of data type *T2* if every value of *T2* is also a value of *T1*. If *T1* and *T2* are not compatible, then *T2* is a *proper supertype* of *T1*. “Compatible” is defined in Subclause 4.12, “Type conversions and mixing of data types”. See also **subtype**.
- rr) **translation**: A method of translating characters in one character repertoire into characters of the same or a different character repertoire.
- ss) **type-preserving function**: An SQL-invoked function, one of whose parameters is a result SQL parameter. The most specific type of the value returned by an invocation of a type-preserving function is identical to the most specific type of the SQL argument value substituted for the result SQL parameter.
- tt) **user-defined type**: A type whose characteristics are specified by a user-defined type descriptor.
- uu) **variable-length**: A characteristic of bit strings and character strings that allows a string to contain any number of bits or characters, respectively, between 0 (zero) and some maximum number, known as the maximum length in bits or characters, respectively, of the string.
- vv) **white space**: Characters used to separate tokens in SQL text; white space may be required (for example, to separate <nondelimiter token>s from one another) and may be used between any two tokens for which there are no rules prohibiting such use. White space is any of the following characters:
- U+0009, Horizontal Tab
 - U+000A, Line Feed
 - U+000B, Vertical Tabulation
 - U+000C, Form Feed
 - U+000D, Carriage Return
 - U+0020, Space
 - U+00A0, No-Break Space
 - U+2000, En Quad
 - U+2001, Em Quad
 - U+2002, En Space
 - U+2003, Em Space
 - U+2004, Three-Per-Em Space
 - U+2005, Four-Per-Em Space
 - U+2006, Six-Per-Em Space
 - U+2007, Figure Space
 - U+2008, Punctuation Space
 - U+2009, Thin Space

3.1 Definitions

- U+200A, Hair Space
- U+200B, Zero Width Space
- U+200C, Zero Width Non-Joiner
- U+200D, Zero Width Joiner
- U+200E, Left-To-Right Mark
- U+200F, Right-To-Left Mark
- U+3000, Ideographic Space
- U+2028, Line Separator
- U+2029, Paragraph Separator
- U+FEFF, Zero Width No-Break Space

NOTE 1 – The notation “U+xyzw” identifies a character position on the Basic Multilingual Plane of ISO/IEC 10646, where each of x, y, z, and w are hexadecimal digits; that character position is in column zw of row xy. In this International Standard, this notation is used only to unambiguously identify characters and is not meant to imply a specific encoding for any implementation’s use of that character.

3.2 Notation

The notation used in this part of ISO/IEC 9075 is defined in ISO/IEC 9075-1.

3.3 Conventions

The conventions used in this part of ISO/IEC 9075 are defined in ISO/IEC 9075-1, with the following additions.

3.3.1 Use of terms

3.3.1.1 Syntactic containment

Let <A> and be syntactic elements; let *A1* be an instance of <A> and let *B1* be an instance of .

A1 *directly contains* *B1* if *A1* contains *B1* without an intervening <set function specification> or <subquery>.

4 Concepts

4.1 Data types

A data type is a set of representable values. Every representable value belongs to at least one data type and some belong to several data types.

Exactly one of the data types of a value V , namely the most specific type of V , is a subtype of every data type of V . A <value expression> E has exactly one declared type, common to every possible result of evaluating E . Items that can be referenced by name, such as SQL parameters, columns, fields, attributes, and variables, also have declared types.

SQL supports three sorts of data types: *predefined data types*, *constructed types*, and *user-defined types*. Predefined data types are sometimes called the “built-in data types”, though not in this International Standard. User-defined data types can be defined by a standard, by an implementation, or by an application.

A constructed type is specified using one of SQL’s data type constructors, ARRAY, REF, and ROW. A constructed type is either an array type, a reference type or a row type, according to whether it is specified with ARRAY, REF, or ROW, respectively. Array types are the only examples of constructed types known generically as collection types.

Every predefined data type is a subtype of itself and of no other data types. It follows that every predefined data type is a supertype of itself and of no other data types. The predefined data types are individually described in each of Subclause 4.2, “Character strings”, through Subclause 4.7, “Datetimes and intervals”.

Row types, reference types and collection types are described in Subclause 4.9, “Row types”, Subclause 4.10, “Reference types”, Subclause 4.11, “Collection types”, respectively.

SQL defines predefined data types named by the following <key word>s: CHARACTER, CHARACTER VARYING, CHARACTER LARGE OBJECT, BINARY LARGE OBJECT, BIT, BIT VARYING, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION, BOOLEAN, DATE, TIME, TIMESTAMP, and INTERVAL. These names are used in the *type designators* that constitute the *type precedence lists* specified in Subclause 9.5, “Type precedence list determination”. In fact, every data type is assigned to exactly one equivalence class named by such a type designator.

For reference purposes:

- The data types CHARACTER, CHARACTER VARYING, and CHARACTER LARGE OBJECT are collectively referred to as *character string types*.
- The data types BIT and BIT VARYING are collectively referred to as *bit string types*.
- The data type BINARY LARGE OBJECT is referred to as the *binary string type* and the values of binary string types are referred to as *binary strings*.
- The data types CHARACTER LARGE OBJECT and BINARY LARGE OBJECT are collectively referred to as *large object string types* and the values of large object string types are referred to as *large object strings*.

4.1 Data types

- Character string types, bit string types, and binary 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 types are referred to as *numbers*.
- The data types TIME WITHOUT TIME ZONE and TIME WITH TIME ZONE are collectively referred to as *time types* (or, for emphasis, as time with or without time zone).
- The data types TIMESTAMP WITHOUT TIME ZONE and TIMESTAMP WITH TIME ZONE are collectively referred to as *timestamp types* (or, for emphasis, as timestamp with or without time zone).
- The data types DATE, time, and timestamp are collectively referred to as *datetime types*.
- Values of datetime types are referred to as *datetimes*.
- The data type INTERVAL is referred to as an *interval type*. Values of interval types are called *intervals*.

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

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

A structured type *ST* is *directly based on* a data type *DT* if *DT* is the declared type of some attribute whose descriptor is included in the descriptor of *ST*.

An array type *AT* is *directly based on* a data type *DT* if *DT* is the element type of *AT*.

A reference type *RT* is *directly based on* a data type *DT* if *DT* is the referenced type.

A row type *RT* is *directly based on* a data type *DT* if *DT* is the declared type of some field (or the data type of the domain of some field) whose descriptor is included in the descriptor of *RT*.

A data type *DT1* is *based on* a data type *DT2* if *DT1* is directly based on *DT2* or *DT1* is directly based on some data type that is based on *DT2*.

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 11.49, "<SQL-invoked routine>", and Subclause 16.1, "<embedded SQL host program>", in ISO/IEC 9075-5. Not every SQL data type has a corresponding data type in every host language.

4.2 Character strings

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

- The name of the specific character string data type (CHARACTER, CHARACTER VARYING, and CHARACTER LARGE OBJECT; NATIONAL CHARACTER, NATIONAL CHARACTER VARYING, and NATIONAL CHARACTER LARGE OBJECT are represented as CHARACTER, CHARACTER VARYING, and CHARACTER LARGE OBJECT, 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.
- 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 defined by implementations, and those defined by applications. Every character set contains the <space> character (equivalent to U+0020). An application defines a character set by assigning a new name to a character set from one of the first two categories. They 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, translations, and form-of-use conversions defined by implementations.

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 (zero) or a positive integer.

All character strings of a given character repertoire are comparable.

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

NOTE 2 – A column may be defined as having a default collating sequence. This default collating sequence for the column may be different from the default collating sequence for its character repertoire, *e.g.*, if the <collate clause> is specified in the column reference. It will be clear from context when the term “default collating sequence” is used whether it is meant for a column or for a character repertoire.

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

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

4.2 Character strings

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

- The name of the character set.
- The name of the default collation for 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 characteristic.

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.

<character overlay function> is a function, OVERLAY, that modifies a string argument by replacing a given substring of the string, which is specified by a given numeric starting position and a given numeric length, with another string (called the replacement string). When the length of the substring is zero, nothing is removed from the original string and the string returned by the function is the result of inserting the replacement string into the original string at the starting position.

<fold> is a pair of functions for converting all the lower case and title case characters in a given string to upper case (UPPER) or all the upper case and title case characters to lower case (LOWER). A lower case character is a character that has the Unicode 'alphabetic' property and whose Unicode name includes 'lower'. An upper case character is a character that has the Unicode 'alphabetic' property and whose Unicode name includes 'upper'. A title case character is a character that has the Unicode 'alphabetic' property and whose Unicode name includes 'title'.

<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 part of ISO/IEC 9075, that S2 be exactly the same sequence of characters as S1, but encoded according some different form-of-use. A typical use might be to convert a character string from two-octet UCS to one-octet Latin1 or *vice versa*.

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

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

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

- The name of the translation,.
- The name of the character set from which it translates.
- The name of the character set to which it translates.
- 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 exact numeric value, 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 (one) for any value of *S2*. If *S1* does not occur in *S2*, then zero is returned. The declared type of a <position expression> is exact numeric.

<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>, <underscore>, or the “escape character” itself is required in the pattern without its special meaning.

<similar predicate> uses the triadic operator SIMILAR (or the inverse, NOT SIMILAR), operating on three character strings and returning a Boolean. SIMILAR determines whether or not a character string “matches” a given “pattern” (also a character string). The pattern is in the form of a “regular expression”. In this regular expression, certain characters (<left bracket>, <right bracket>, <left paren>, <right paren>, <vertical bar>, <circumflex>, <minus sign>, <plus sign>, <asterisk>, <underscore>, <percent>) have a special meaning. The optional third argument specifies the “escape character”, for use when one of the special characters or the “escape character” itself is required in the pattern without its special meaning.

4.2.2.3 Operations involving large object character strings

Large object strings cannot be operated on by all string operations. Large object strings can, however, be operated on by the following operations:

- <null predicate>.
- <like predicate>.
- <similar predicate>.
- <position expression>.
- <comparison predicate> with an <equals operator> or <not equals operator>.
- <quantified comparison predicate> with the <equals operator> or <not equals operator>.

4.2 Character strings

As a result of these restrictions, large object strings and large object string columns cannot be referenced in (among other places):

- Predicates other than those listed above and the <exists predicate>.
- <general set function>.
- <group by clause>.
- <order by clause>.
- <unique constraint definition>.
- <referential constraint definition>.
- <select list> of a <query specification> that has a <set quantifier> of DISTINCT.
- UNION, INTERSECT, and EXCEPT.
- Columns used for matching when forming a <joined table>.

All the operations described within Subclause 4.2.2.1, “Operators that operate on character strings and return character strings”, and Subclause 4.2.2.2, “Other operators involving character strings”, are supported for large object character strings.

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 3 – The default collating sequence for a character repertoire is defined in Subclause 10.6, “<character set specification>”, and Subclause 11.30, “<character set definition>”.
- When one or more columns are involved (*e.g.*, comparing two columns, or comparing a column to a literal), then provided that all columns involved have the same default collating sequence and there is no explicit specification of a collating sequence, that default collating sequence is used.
- When columns are involved having different default collating sequences, explicit specification of the collating sequence in the expression is required via the <collate clause>.
- 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 characteristic. This characteristic has the values *Coercible*, *Implicit*, *No collating sequence*, and *Explicit*. <character value expression>s with the *Coercible*, *Implicit*, or *Explicit* characteristics have a collating sequence.

A <character value expression> consisting of a column reference has the coercibility characteristic *Implicit*, 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 coercibility characteristic *Coercible*, with the default collation for its character repertoire. A <character value expression> simply containing a <collate clause> has the coercibility characteristic *Explicit*, with the collating sequence specified in the <collate clause>.

NOTE 4 – When the coercibility characteristic 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 characteristic 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	
No collating sequence		Explicit	Y	Explicit	Y
No collating sequence		Any, except Explicit	Any	No collating sequence	
Explicit	X	Explicit	X	Explicit	X
Explicit	X	Coercible	default	Explicit	X
Explicit	X	Implicit	Y	Explicit	X
Explicit	X	No collating sequence		Explicit	X
Explicit	X	Explicit	X	Explicit	X
Explicit	X	Explicit	$Y \neq X$	Not permitted: <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>
Implicit	X	Explicit	Y	Y
No collating sequence		Any except Explicit	Any	Not permitted: <i>invalid syntax</i>
No collating sequence		Explicit	X	X
Explicit	X	Coercible	default	X
Explicit	X	Implicit	Y	X
Explicit	X	No collating sequence		X
Explicit	X	Explicit	X	X
Explicit	X	Explicit	$Y \neq X$	Not permitted: <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.

4.2.4 Named character sets

An SQL <character set specification> allows a reference to a character set name defined by a standard, an implementation, or a user. The following SQL supported character set names are specified as part of ISO/IEC 9075:

- SQL_CHARACTER specifies the name of a character repertoire that consists of the 88 <SQL language character>s as specified in Subclause 5.1, “<SQL terminal character>”. It consists of the 52 uppercase and lowercase simple latin characters, 10 digits, and 26 <SQL special character>s, including: <space>, <double quote>, <percent>, <ampersand>, <quote>, <left paren>, <right paren>, <asterisk>, <plus sign>, <comma>, <minus sign>, <period>, <solidus>, <colon>, <semicolon>, <less than operator>, <equals operator>, <greater than operator>, <question mark>, <underscore>, <vertical bar>, <left bracket>, <right bracket>, <circumflex>, <left brace>, and <right brace>. The 88 characters specified as <SQL language character>s are all included in the ISO International Reference Version (IRV) characters specified in ISO 646:1991. The characters in IRV are included in many other international character set definitions. In addition, 82 of these characters (all except <vertical bar>, <left bracket>, <right bracket>, <circumflex>, <left brace>, and <right brace>) are in the most stable subset of IRV that is by ISO

convention, is included in every latin-based ISO standard set of characters. As far as can be determined, <vertical bar>, <left bracket>, <right bracket>, <circumflex>, <left brace>, and <right brace> are included in most character sets that enjoy world-wide use. Thus, the SQL_CHARACTER repertoire is the most universal of the character sets named herein. The collation and form-of-use of SQL_CHARACTER is implementation-defined.

- GRAPHIC_IRV (or ASCII_GRAPHIC) specifies the name of a character repertoire that consists of the 95-character graphic subset of the International Reference Version (IRV) as specified in ISO 646:1991. The form-of-use is that corresponding to the coded representation of each character by a single byte (possibly 7-bit, 8-bit, or other), with no designation escape sequences for other character sets. The default collating sequence is that corresponding to the bit combinations defined by ISO 646:1991. The GRAPHIC_IRV character set is a superset of the <SQL language character>s. The 7 characters included in GRAPHIC_IRV that are not <SQL language character>s are, in collation order: Exclamation mark (!), Number sign (#), Dollar sign (\$), Commercial at (@), Reverse solidus (\), Grave accent (`), and Tilde (~). Of these 7 characters, only “!” is in the most stable subset of ISO 646, whereas “#” competes with the British pound sign, “\$” competes with the international currency symbol, and the others occupy positions in ISO 646 that are reserved for national or application-oriented use. However, all are the default IRV values specified in ISO 646 when no national or application-specific version is explicitly specified.
- LATIN1 specifies the name of a character repertoire that consists of the 191 graphic characters defined in ISO 8859-1. The form-of-use is that corresponding to the coded representation of each character by a single 8-bit byte, with no designation escape sequences for other character sets. The default collating sequence is that corresponding to the bit combinations defined by ISO 8859-1. The LATIN1 character set is a superset of GRAPHIC_IRV and, when restricted to the GRAPHIC_IRV characters, produces the same collation as GRAPHIC_IRV. LATIN1 consists of all characters commonly used in the following languages: Danish, Dutch, English, Faeroese, Finnish, French, German, Icelandic, Irish, Italian, Norwegian, Portuguese, Spanish, and Swedish. It also includes the following special symbols, in collation order: No-break space, Inverted exclamation mark (¡), Cent sign (¢), Pound sign (£), Currency sign (₹), Yen sign (¥), Broken bar (|), Paragraph sign (¶), Diaeresis (¨), Copyright sign (©), Feminine ordinal indicator (ª), Left angle quotation mark («), Not sign (¬), Soft hyphen, Registered trade mark sign (®), Macron (ˆ), Degree sign (°), Plus-minus sign (±), Superscript two (²), Superscript three (³), Acute accent (´), Micro sign (µ), Pilcrow sign, Middle dot (·), Cedilla (¸), Superscript one (¹), Masculine ordinal indicator (º), Right angle quotation mark (»), Fraction one quarter (1/4), Fraction one half (1/2), Fraction three quarters (3/4), and Inverted question mark (¿). Other characters include the Multiplication sign and the Division sign. In LATIN1, all GRAPHIC_IRV characters precede the non-GRAPHIC_IRV characters in the default collation, followed by the special symbols, followed by the accented capital letters, followed by the accented small letters. The Multiplication sign (×) is in the middle of the accented capital letters and the Division sign (÷) is in the middle of the accented small letters.
- LATIN1 is subject to the following conformance requirements, as specified in Clause 3, "Conformance", of ISO 8859-1:
 - A set of graphic characters is in conformance with ISO 8859-1 if it comprises all graphic characters specified therein to the exclusion of any other and if their coded representations are those specified by ISO 8859-1.
 - Equipment claimed to implement ISO 8859-1 shall implement all 191 characters.

4.2 Character strings

- ISO8BIT (or ASCII_FULL) specifies the name of a character repertoire that consists of all 255 characters, each consisting of exactly 8 bits, as specified in ISO 4873 and ISO 8859-1, including all control characters and all graphic characters except the character corresponding to the numeric value 0 (zero). The form-of-use is that corresponding to the coded representation of each character by a single 8-bit byte, with no designation escape sequences for other character sets. The default collating sequence is that corresponding to the bit combinations defined. The ISO8BIT character set is a superset of LATIN1 and, when restricted to the LATIN1 characters, produces the same collation and form-of-use.
- UTF16 and ISO10646 specify the name of a character repertoire that consists of every character represented by The Unicode Standard Version 2.0 and by ISO/IEC 10646 UTF-16, where each character is encoded using the UTF-16 encoding, occupying either 1 (one) or 2 octets.
- UTF8 specifies the name of a character repertoire that consists of every character represented by The Unicode Standard Version 2.0 and by ISO/IEC 10646 UTF-8, where each character is encoded using the UTF-8 encoding, occupying from 1 (one) through 6 octets.
- UCS2 specifies the name of a character repertoire that consists of every character represented by The Unicode Standard Version 2.0 and by ISO/IEC 10646 UCS2, where each character is encoded using the UCS2 encoding, in which each character occupies exactly 2 octets.
- SQL_TEXT specifies the name of a character repertoire that includes the <SQL language character>s and all other characters that are in character sets supported by the implementation. The SQL_TEXT character set is a superset of SQL_CHARACTER. The collation and form-of-use of SQL_TEXT is implementation-defined.
- SQL_IDENTIFIER specifies the name of a character repertoire that includes the <SQL language character>s and all other characters that the SQL-implementation supports for use in <regular identifier>s, which is the same as the repertoire that the SQL-implementation supports for use in <delimited identifier>s. The SQL_IDENTIFIER character set is a superset of SQL_CHARACTER but may be a subset of SQL_TEXT. The collation and form-of-use of SQL_IDENTIFIER is implementation-defined.
- The character sets SQL_CHARACTER, GRAPHIC_IRV (or ASCII_GRAPHIC), LATIN1, ISO8BIT (or ASCII_FULL), and UNICODE (or ISO10646) have both a “floor” and “ceiling” requirement to consist of exactly the characters specified. Any character data type associated with one of these character sets has an implied integrity constraint limiting a value of the data type to be a character string consisting only of characters from the specified character set. The SQL_TEXT and SQL_IDENTIFIER character sets have a similar “floor” requirement in that they must contain all characters that are in other character sets supported by the implementation (for SQL-data and for <identifier>s, respectively); however, SQL_TEXT and SQL_IDENTIFIER do not have a “ceiling” requirement.

4.3 Binary strings

A binary string is a sequence of octets that does not have either a character set or collation associated with it.

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

- The name of the data type (BINARY LARGE OBJECT).

- The maximum length of the binary string data type (in octets).

4.3.1 Binary string comparison

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

4.3.2 Operations involving binary strings

4.3.2.1 Operators that operate on binary strings and return binary strings

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

<blob substring function> is a triadic function identical in syntax and semantics to <character substring function> except that the returned value is a binary string.

<blob overlay function> is a function identical in syntax and semantics to <character overlay function> except that the first argument, second argument, and returned value are all binary strings.

<trim function> when applied to binary strings is identical in syntax (apart from the default <trim character>) and semantics to the corresponding operation on character strings except that the returned value is a binary string.

4.3.2.2 Other operators involving binary strings

<length expression> returns the length of a given binary string, as an exact numeric value, in characters, octets, or bits according to the choice of function.

<position expression> when applied to binary strings is identical in syntax and semantics to the corresponding operation on character strings except that the operands are binary strings.

<like predicate> when applied to binary strings is identical in syntax and semantics to the corresponding operation on character strings except that the operands are binary strings.

4.4 Bit strings

A bit string is a sequence of bits, each having the value of 0 (zero) or 1 (one). A bit string has a length, which is the number of bits in the string. The length is 0 (zero) 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).
- The length of the bit string data type (in bits).

4.4 Bit strings

4.4.1 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>”.

4.4.2 Operations involving bit strings

4.4.2.1 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.

4.4.2.2 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 (one) for any value of *S2*. If *S1* does not occur in *S2*, then zero is returned.

4.5 Numbers

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

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

- The name of the specific numeric data type (NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, or DOUBLE PRECISION).
- The precision of the numeric data type.
- The scale of the numeric data type, if it is an exact numeric data type.
- An indication of whether the precision (and scale) are expressed in decimal or binary terms.

A value described by a numeric data type descriptor is always signed.

4.5.1 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 (zero) indicates that the number is an integer. For a scale of *S*, the exact numeric value is the integer value of the significant digits multiplied by 10^{-S} .

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

Whenever an exact or approximate numeric value is assigned to an exact numeric value site, an approximation of its value that preserves leading significant digits after rounding or truncating is represented in the declared 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 numeric value N for an <exact numeric type> T is a value V in T such that N is not closer to zero than is V and there is no value in T between V and N .

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

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

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

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

Whenever an exact or approximate numeric value is assigned to an approximate numeric value site, an approximation of its value is represented in the declared 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 6.26, "<numeric value expression>".

4.5.2 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 4.4.2, "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 4.4.2, "Operations involving bit strings") operates on a string argument and returns an integer.
- <extract expression> (see Subclause 4.7.3, "Operations involving datetimes and intervals") operates on a datetime or interval argument and returns an integer.
- <cardinality expression> (see Subclause 4.11.3, "Operations involving collections") operates on a collection argument and returns an integer.

4.5 Numbers

- <absolute value expression> operates on a numeric argument and returns its absolute value in the same most specific type.
- <modulus expression> operates on two exact numeric arguments with scale 0 (zero) and returns the modulus (remainder) of the first argument divided by the second argument as an exact numeric with scale 0 (zero).

4.6 Boolean types

The data type boolean comprises the distinct truth values *true* and *false*. Unless prohibited by a NOT NULL constraint, the boolean data type also supports the *unknown* truth value as the null value. This specification does not make a distinction between the null value of the boolean data type and the *unknown* truth value that is the result of an SQL <predicate>, <search condition>, or <boolean value expression>; they may be used interchangeably to mean exactly the same thing.

The boolean data type is described by the boolean data type descriptor. The boolean data type descriptor contains:

- The name of the boolean data type (BOOLEAN).

4.6.1 Comparison and assignment of booleans

All boolean data type values and SQL truth values are mutually comparable and assignable. The value *true* is greater than the value *false*, and any comparison involving the null value or an *unknown* truth value will return an *unknown* result. The values *true* and *false* may be assigned to any site having a boolean data type; assignment of *unknown*, or the null value, is subject to the nullability characteristic of the target.

4.6.2 Operations involving booleans

4.6.2.1 Operations on booleans that return booleans

The monadic boolean operator NOT and the dyadic boolean operators AND and OR take boolean operands and produce a boolean result (see Table 13, “Truth table for the AND boolean operator”, and Table 14, “Truth table for the OR boolean operator”).

4.6.2.2 Other operators involving booleans

Every SQL <predicate>, <search condition>, and <boolean value expression> may be considered as an operator that returns a boolean result.

4.7 Datetimes and intervals

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

A datetime data type descriptor contains:

- The name of the specific datetime data type (DATE, TIME WITHOUT TIME ZONE, TIMESTAMP WITHOUT TIME ZONE, TIME WITH TIME ZONE, or TIMESTAMP WITH TIME ZONE).

- The value of the <time fractional seconds precision>, if it is a TIME WITHOUT TIME ZONE, TIMESTAMP WITHOUT TIME ZONE, 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.
- The <interval qualifier> that describes the precision of the interval data type.

A value described by an interval data type descriptor is always signed.

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.

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

4.7.1 Datetimes

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

Table 4—Fields in datetime values

Keyword	Meaning
Primary datetime fields	
YEAR	Year
MONTH	Month within year
DAY	Day within month
HOURL	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 <primary datetime field>s. This is, from most significant to least significant: YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.

The <primary 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 part of ISO/IEC 9075:

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

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

A datetime data type that specifies WITH TIME ZONE is a data type that is *datetime with time zone*, while a datetime data type that specifies WITHOUT TIME ZONE is a data type that is *datetime without time zone*.

The surface of the earth is divided into zones, called time zones, in which every correct clock tells the same time, known as *local time*. Local time is equal to UTC (Coördinated Universal Time) plus the *time zone displacement*, which is an interval value that ranges between INTERVAL –'12:59' HOUR TO MINUTE and INTERVAL +'13:00' HOUR TO MINUTE. The time zone displacement is constant throughout a time zone, changing at the beginning and end of Daylight Time, where applicable.

A datetime value, of data type TIME WITHOUT TIME ZONE or TIMESTAMP WITHOUT TIME ZONE, may represent a local time, whereas a datetime value of data type TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE represents UTC. 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. Whether an SQL-implementation supports leap seconds, and the consequences of such support for date and interval arithmetic, is implementation-defined.

For the convenience of users, whenever a datetime value with time zone is to be implicitly derived from one without (for example, in a simple assignment operation), SQL assumes the value without time zone to be local, subtracts the default SQL-session time zone displacement from it to give UTC, and associates that time zone displacement with the result.

Conversely, whenever a datetime value without time zone is to be implicitly derived from one with, SQL assumes the value with time zone to be UTC, adds the time zone displacement to it to give local time, and the result, without any time zone displacement, is local.

The preceding principles, as implemented by <cast specification>, result in data type conversions between the various datetime data types, as summarized in Table 5, "Datetime data type conversions".

Table 5—Datetime data type conversions

	to DATE	to TIME WITHOUT TIME ZONE	to TIME WITH TIME ZONE	to TIMESTAMP WITHOUT TIME ZONE	to TIMESTAMP WITH TIME ZONE
from DATE	<i>trivial</i>	<i>not sup- ported</i>	<i>not sup- ported</i>	Copy year, month, and day; set hour, minute, and second to 0 (zero)	$SV \Rightarrow TS_{w/oTZ}$ $\Rightarrow TS_{w/TZ}$
from TIME WITHOUT TIME ZONE	<i>not sup- ported</i>	<i>trivial</i>	$TV.UTC =$ $SV - STZD$ (modulo 24); $TV.TZ =$ $STZD$	Copy date fields from CURRENT_ DATE and time fields from TZ	$SV \Rightarrow TS_{w/oTZ}$ $\Rightarrow TS_{w/TZ}$
from TIME WITH TIME ZONE	<i>not sup- ported</i>	$SV.UTC +$ $SV.TZ$ (module 24)	<i>trivial</i>	$SV \Rightarrow TS_{w/oTZ}$ $\Rightarrow TS_{w/TZ}$	Copy date fields from CURRENT_ DATE and time and time zone fields from SV
from TIMESTAMP WITHOUT TIME ZONE	Copy date fields from SV	Copy time fields from TZ	$SV \Rightarrow$ $TS_{w/TZ}$ $\Rightarrow TS_{w/oTZ}$	<i>trivial</i>	$TV.UTC = SV$ $- STZD;$ $TV.TZ = STZD$
from TIMESTAMP WITH TIME ZONE	$SV \Rightarrow$ $TS_{w/oTZ}$ $\Rightarrow DATE$	$SV \Rightarrow$ $TS_{w/oTZ}$ $\Rightarrow TIME_{w/oTZ}$	Copy time and time zone fields from SV	$SV.UTC +$ $SV.TZ$	<i>trivial</i>

4.7.2 Intervals

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

Table 6, “Fields in year-month INTERVAL values”, 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 6—Fields in year-month INTERVAL values

Keyword	Meaning
YEAR	Years
MONTH	Months

Table 7, “Fields in day-time INTERVAL values”, 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 7—Fields in day-time INTERVAL values

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

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

Within a value of type interval, the first field is constrained only by the <interval leading field precision> of the associated <interval qualifier>. Table 8, “Valid values for fields in INTERVAL values”, specifies the constraints on subsequent field values.

Table 8—Valid values for fields in INTERVAL values

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 and have precision 2 when not the first field. 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. When not the first field, SECOND has a precision of 2 places before the decimal point.

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 <primary datetime field>s to either the most significant end of one interval, the least significant end of one interval, or both. New least significant <primary datetime field>s are assigned a value of 0 (zero). 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.

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 <primary datetime field>s to either the most significant end of one interval or the least significant end of one interval, or both. New least significant <primary datetime field>s are assigned a value of 0 (zero). 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.

4.7.3 Operations involving datetimes and intervals

Table 9, “Valid operators involving datetimes and intervals”, specifies the declared types of arithmetic expressions involving datetime and interval operands.

Table 9—Valid operators involving datetimes and intervals

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

Arithmetic operations involving values 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 values of type datetime require that the datetime values be mutually comparable. Operations involving values of type interval require that the interval values 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.

<interval absolute value expression> operates on an interval argument and returns its absolute value in the same most specific type.

4.8 User-defined types

4.8 User-defined types

A user-defined type is a schema object, identified by a <user-defined type name>. The definition of a user-defined type specifies a number of components, including in particular a list of attribute definitions. Although the attribute definitions are said to define the representation of the user-defined type, in fact they implicitly define certain functions (observers and mutators) that are part of the interface of the user-defined type; physical representations of user-defined type values are implementation-dependent.

The representation of a user-defined type is expressed either as a single data type (some predefined data type, called the *source type*), in which case the user-defined type is said to be a *distinct type*, or as a list of attribute definitions, in which case it is said to be a *structured type*.

The definition of a user-defined type may include a <method specification list> consisting of one or more <method specification>s. A <method specification> is either an <original method specification> or an <overriding method specification>. Each <original method specification> specifies the <method name>, the <specific name>, the <SQL parameter declaration list>, the <returns data type>, the <result cast from type> (if any), the <transform group specification> (if any), whether the method is type-preserving, the <language clause>, the <parameter style> if the language is not SQL, whether STATIC is specified, whether the method is deterministic, to what extent the method accesses SQL (possibly writes SQL data, possibly reads SQL data, possibly contains SQL, or does not possibly contain SQL), and whether the method should be evaluated as NULL whenever any argument is NULL, without actually invoking the method.

Each <overriding method specification> specifies the <method name>, the <specific name>, the <SQL parameter declaration list> and the <returns data type>. For each <overriding method specification>, there must be an <original method specification> with the same <method name> and <SQL parameter declaration list> in some proper supertype of the user-defined type. Every SQL-invoked method in a schema must correspond to exactly one <original method specification> or <overriding method specification> associated with some user-defined type existing in that schema.

A method *M* that corresponds to an <original method specification> in the definition of a structured type *T1* is an *original method* of *T1*. A method *M* that corresponds to an <overriding method specification> in the definition of *T1* is an *overriding method* of *T1*.

A method *M* is a *method of type T1* if one of the following holds:

- *M* is an original method of *T1*.
- *M* is an overriding method of *T1*.
- There is a proper supertype *T2* of *T1* such that *M* is an original or overriding method of *T2* and such that there is no method *M3* such that *M3* has the same <method name> and <SQL parameter declaration list> as *M* and *M3* is an original method or overriding method of a type *T3* such that *T2* is a proper supertype of *T3* and *T3* is a supertype of *T1*.

If *T1* is a subtype of *T2* and *M1* is a method of *T1* such that there exists a method *M2* of *T2* such that *M1* and *M2* have the same <method name> and the same unaugmented <SQL parameter declaration list>, then *M1* is an *inherited method* of *T1* from *T2*.

A user-defined type is described by a user-defined type descriptor. A user-defined type descriptor contains:

- The name of the user-defined type. This is the type designator of that type, used in type precedence lists (see Subclause 9.5, “Type precedence list determination”).