INTERNATIONAL STANDARD

ISO/IEC 9075-1

Sixth edition
2023-06

Information technology — Database languages — SQL —

Part 1: Framework (SQL/Framework)

Technologies de l'information — Langages de base de données — SQL —

Partie 1: Charpente (SQL/Charpente)

Reference number
ISO/IEC 9075-1:2023(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Tables

Table

Page

Figures

Figure **Page**

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 technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC have not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

This sixth edition cancels and replaces the fifth edition (ISO/IEC 9075-1:2016), which has been technically revised. It also incorporates the Technical Corrigendum ISO/IEC 9075-1:2016/Cor.1:2022.

The main changes are as follows:

— addition and refinement of the terms and concepts to support new data types required by incremental parts;

— clarification and correction of the merge instructions for Technical Corrigenda and incremental parts;

— improve the presentation and accuracy of the summaries of implementation-defined and implementation-dependent aspects of this document;

— introduction of several digital artifacts;

— alignment with updated ISO house style and other guidelines for creating standards.

This sixth edition of ISO/IEC 9075-1 is designed to be used in conjunction with the following editions of other parts of the ISO/IEC 9075 series, all published 2023:

— ISO/IEC 9075-2, sixth edition;

— ISO/IEC 9075-3, sixth edition;

— ISO/IEC 9075-4, seventh edition;

— ISO/IEC 9075-9, fifth edition;

— ISO/IEC 9075-10, fifth edition;

— ISO/IEC 9075-11, fifth edition;

— ISO/IEC 9075-13, fifth edition;

— ISO/IEC 9075-14, sixth edition;

— ISO/IEC 9075-15, second edition;

— ISO/IEC 9075-16, first edition.

A list of all parts in the ISO/IEC 9075 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

The organization of this document is as follows:

1) Clause 1, "Scope", specifies the scope of this document.

2) Clause 2, "Normative references", identifies additional standards that, through reference in this document, constitute provisions of the ISO/IEC 9075 series.

3) Clause 3, "Terms and definitions", defines terms and definitions used in this document and in other parts of the ISO/IEC 9075 series.

4) Clause 4, "Concepts", describes the concepts used in the ISO/IEC 9075 series.

5) Clause 5, "Parts of the ISO/IEC 9075 series", summarizes the content of each of the parts of the ISO/IEC 9075 series, in terms of the concepts described in Clause 4, "Concepts".

6) Clause 6, "Notation and conventions used in other parts of the ISO/IEC 9075 series", defines notation and conventions used in other parts of the ISO/IEC 9075 series.

7) Clause 7, "Annexes to the parts of the ISO/IEC 9075 series", describes the content of annexes of other parts of the ISO/IEC 9075 series.

8) Clause 8, "Status codes", defines values that identify the status of the execution of SQL-statements and the mechanisms by which those values are returned.

9) Clause 9, "Conformance", specifies requirements that apply to claims of conformance to all or some of the parts of the ISO/IEC 9075 series.

10) Annex A, "SQL conformance summary", is an informative Annex. It summarizes the conformance requirements of the SQL language.

11) Annex B, "Implementation-defined elements", is an informative Annex. It lists those features for which the body of this document states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or other aspect is partly or wholly implementation-defined.

12) Annex C, "Implementation-dependent elements", is an informative Annex. It lists those features for which the body of this document states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or other aspect is partly or wholly implementation-dependent.

13) Annex D, "SQL optional feature taxonomy", is an informative Annex. It identifies features of the SQL language specified in this document by an identifier and a short descriptive name. This taxonomy is used to specify conformance.

14) Annex E, "Deprecated features", is an informative Annex. It lists features that the responsible Technical Committee intends not to include in a future edition of this document.

15) Annex F, "Incompatibilities with ISO/IEC 9075:2016", is an informative Annex. It lists incompatibilities with the previous edition of this document.

16) Annex G, "Defect Reports not addressed in this edition of this document", is an informative Annex. It describes the Defect Reports that were known at the time of publication of this document. Each of these problems is a problem carried forward from the previous edition of this document. No new problems have been created in the drafting of this edition of this document.

17) Annex H, "Maintenance and interpretation of SQL", is an informative Annex. It describes the formal procedures for maintenance and interpretation of the ISO/IEC 9075 series.

18) Annex I, "Support for the use of inter-document links in the ISO/IEC 9075 series", is an informative Annex. It describes conventions that users of the ISO/IEC 9075 series use in order to maximize the facilities provided in the parts of the ISO/IEC 9075 series.

Information technology — Database language SQL —

Part 1:
Framework (SQL/Framework)

1 Scope

This document describes the conceptual framework used in other parts of the ISO/IEC 9075 series to specify the grammar of SQL and the result of processing statements in that language by an SQL-implementation.

This document also defines terms and notation used in the other parts of the ISO/IEC 9075 series.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

ISO/IEC 9075-2, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*

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

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

ISO/IEC 9075-9, *Information technology — Database languages — SQL — Part 9: Management of External Data (SQL/MED)*

ISO/IEC 9075-10, *Information technology — Database languages — SQL — Part 10: Object Language Bindings (SQL/OLB)*

ISO/IEC 9075-11, *Information technology — Database languages — SQL — Part 11: Information and Definition Schemas (SQL/Schemata)*

ISO/IEC 9075-13, *Information technology — Database languages — SQL — Part 13: SQL Routines and Types Using the Java™ Programming Language (SQL/JRT)*

ISO/IEC 9075-14, *Information technology — Database languages — SQL — Part 14: XML-Related Specifications (SQL/XML)*

ISO/IEC 9075-15, *Information technology — Database languages — SQL — Part 15: Multidimensional Arrays (SQL/MDA)*

ISO/IEC 9075-16, *Information technology — Database languages — SQL — Part 16: Property Graph Queries (SQL/PGQ)*

ISO/IEC 10646:2020, *Information technology — Universal Multi-Octet Coded Character Set (UCS)*

ISO/IEC 14651, *Information technology — International string ordering and comparison — Method for comparing character strings and description of the common template tailorable ordering*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1 Definitions taken from ISO/IEC 10646:2020

3.1.1
character
member of a set of elements used for the organization, control, or representation of textual data

Note 1 to entry: This is identical to *abstract character* as defined by The Unicode® Standard. In this document, when the relevant character repertoire is UCS, a character can be thought of as *that which is represented by one code point*.

[SOURCE: ISO/IEC 10646:2020, 3.5]

3.2 Definitions provided in this document

3.2.1
atomic, adj.
incapable of being subdivided

3.2.2
column
field of the row type of a table

3.2.3
compilation unit
segment of executable code, possibly consisting of one or more subprograms

3.2.4
data type
set of representable values

3.2.5
descriptor
coded description of the metadata of an SQL object

3.2.6
field
(field name, data type) pair

Note 1 to entry: A value of a field is a value of its data type.

3.2.7
identifier
object by which something is identified

3.2.8
identify, verb
to reference something without ambiguity

3.2.9
implementation-defined, adj.
possibly differing between SQL-implementations, but specified by the implementor for each particular SQL-implementation

3.2.10
implementation-dependent, adj.
possibly differing between SQL-implementations, but not specified by the ISO/IEC 9075 series, and not required to be specified by the implementor for a particular SQL-implementation

3.2.11
instance
<of a value> physical representation of a value

Note 1 to entry: Each instance is at exactly one site. An instance has a data type that is the data type of its value.

3.2.12
null value
special value that is used to indicate the absence of any data value

3.2.13
object
<as in "x object"> *thing* that is a component of, or is otherwise associated with, some x and cannot exist independently of that x

Note 1 to entry: For example, an SQL object is an object that exists only in the context of SQL; an SQL-schema object is an object that exists in some SQL-schema.

3.2.14
persistent, adj.
continuing to exist indefinitely, until destroyed deliberately

Note 1 to entry: Referential and cascaded actions are regarded as deliberate. Actions incidental to the termination of an SQL-transaction or an SQL-session are not regarded as deliberate.

3.2.15
property
<of an object> attribute, quality, or characteristic of the object

3.2.16
row
value of a row type, i.e., a sequence of field values; for rows in a table, a sequence of column values

3.2.17
scope
<of a standard> clause in the standard that defines the subject of the standard and the aspects covered, thereby indicating the limits of applicability of the standard or of particular parts of it

3.2.18
scope
<of a name or declaration> one or more BNF non-terminal symbols within which the name or declaration is effective

3.2.19
scope
<of a reference type> table that a value of that reference type references

3.2.20
sequence
ordered collection of objects that are not necessarily distinct

3.2.21
site
place occupied by an instance of a value of some specified data type (or subtype of that data type)

3.2.22
SQL-connection
association between an SQL-client and an SQL-server

3.2.23
SQL-environment
context in which SQL-data exists and SQL-statements are executed

3.2.24
SQL-implementation
processor that processes SQL-statements

Note 1 to entry: A *conforming SQL-implementation* is an SQL-implementation that satisfies the requirements for SQL-implementations as defined in Clause 9, "Conformance".

3.2.25
SQL-session
context within which a single user, from a single SQL-agent, executes a sequence of consecutive SQL-statements over a single SQL-connection

3.2.26
SQL-statement
string of characters that conforms to the Format and Syntax Rules specified in the parts of the ISO/IEC 9075 series

3.2.27
table
collection of zero or more rows

3.2.28
whitespace
sequence of one or more characters that have the Unicode property White_Space

4 Concepts

4.1 What is SQL?

The name "SQL" is correctly pronounced "ess cue ell," instead of the somewhat common "sequel". SQL, specified in the parts of the ISO/IEC 9075 series, is a *database language* (more precisely, a *data sublanguage*) used for access to pseudo-relational databases that are managed by pseudo-relational database management systems (RDBMS).

NOTE 1 — Many books and articles "define" SQL by parenthetically claiming that the letters stand for "Structured Query Language". While this was undoubtedly true for the original prototypes, it is not true of the standard. When the letters appear in product names, they have often been assigned this meaning by the product implementors, but users are ill-served by claims that the word "structured" accurately describes the language. In the SQL standard, the letters do not stand for anything at all. There are afficionados who assert that the letters stand for "SQL Query Language", a recursive acronym in the tradition of GNU (GNU's Not Unix) and SPARQL (SPARQL Protocol And RDF Query Language).

SQL is *based on*, but is not a strict implementation of, the relational model of data Relational Model, making SQL "pseudo-relational" instead of truly relational. There are two principal ways in which SQL breaks from the relational model.

— The relational model requires that every *relation*[1] have no duplicate rows (that is, rows for which all corresponding attributes have equal values). A consequence of this requirement is that every relation has a primary key. SQL does not enforce this requirement, but provides syntax that permits application developers to enforce the requirement as needed.

— The relational model does not specify or recognize any sort of *flag* or other marker that represents unspecified, unknown, or otherwise missing data values. Consequently, the relational model depends only on two-valued (true/false) logic. SQL provides a "null value" that serves this purpose. In support of null values, SQL also depends on three-valued (true/false/unknown) logic.

The SQL standard defines two distinct aspects of the language: a *data definition language* (known in the standard as a "schema manipulation language" that supports the creation and destruction of database objects, such as tables, columns, and integrity constraints; and a *data manipulation language* that allows the retrieval, creation, modification, and deletion of data.

An especially important characteristic of SQL is that it is *declarative* (as opposed to *procedural*) in nature.[2] That is, SQL's syntax and semantics allow application writers to specify the rules by which results may be computed by an SQL system, rather than specifying the algorithms by which those results must be computed. This contrasts with earlier database languages that required the details specification of step-by-step algorithms to retrieve, filter, and format the data required by applications.

4.2 Use of terms

The concepts on which the ISO/IEC 9075 series is based are described in terms of objects, in the usual sense of the word.

Some objects are a component of an object on which they depend. If an object ceases to exist, then every object dependent on that objects also ceases to exist. The metadata representation of an object is known as a descriptor. See also Subclause 6.3.4, "Descriptors".

1 Terms defined by the relational model are used correctly in this Subclause, but are not defined or explained in this document; readers are referred to the formal specification of the relational model.

2 ISO/IEC 9075-4, specifies a number of SQL statements that provide a *procedural* capability through which procedures and functions can be written entirely in SQL. Procedural programs can also be written in ordinary "host" programming languages with embedded SQL statements.

4.3 Caveat

This Clause describes concepts that are, for the most part, specified precisely in other parts of the ISO/IEC 9075 series. In any case of discrepancy, the specification in the other part is to be presumed correct.

4.4 SQL-environments and their components

4.4.1 SQL-environments

An SQL-environment comprises:

— one SQL-agent;

— one SQL-implementation;

— zero or more SQL-client modules, containing externally-invoked procedures available to the SQL-agent;

— zero or more authorization identifiers;

— zero or more user mappings;

— zero or more routine mappings;

— zero or more catalogs, each of which contains one or more SQL-schemas, zero or more foreign server descriptors, and zero or more foreign-data wrapper descriptors;

— the sites, principally base tables, that contain SQL-data, as described by the contents of the schemas; this data may be thought of as "the database", but the term is not used in the ISO/IEC 9075 series, because it has different meanings in the more general context.

4.4.2 SQL-agents

An *SQL-agent* is that which causes the execution of SQL-statements. In the case of the direct invocation of SQL (see Subclause 5.3.2.4, "Direct invocation of SQL"), it is implementation-defined (IW002). Alternatively, it may consist of one or more compilation units that, when executed, invoke externally-invoked procedures in an SQL-client module.

4.4.3 SQL-implementations

4.4.3.1 Introduction to SQL-implementations

An *SQL-implementation* is a processor that executes SQL-statements, as required by the SQL-agent. An SQL-implementation, as perceived by the SQL-agent, includes one SQL-client, to which that SQL-agent is bound, and one or more SQL-servers. An SQL-implementation is able to conform to ISO/IEC 9075 without allowing more than one SQL-server to exist in an SQL-environment.

Because an SQL-implementation can be specified only in terms of how it executes SQL-statements, the concept denotes an installed instance of some software (database management system). ISO/IEC 9075 does not distinguish between features of the SQL-implementation that are determined by the software implementor and those determined by the installer.

ISO/IEC 9075 recognizes that SQL-client and SQL-server software may have been obtained from different implementors; it does not specify the method of communication between SQL-client and SQL-server.

4.4.3.2 SQL-clients

An *SQL-client* is a processor, perceived by the SQL-agent to be part of the SQL-implementation, that establishes SQL-connections between itself and SQL-servers and maintains a diagnostics area and other state data relating to interactions between itself, the SQL-agent, and the SQL-servers.

When one or more SQL-connections have been established for an SQL-client, the diagnostics area maintained by that SQL-client is a copy of the first diagnostics area in the diagnostics area stack (see Subclause 4.10.5, "Diagnostic information") maintained by the SQL-server of the current SQL-connection. When no SQL-connection exists, the diagnostics area is either empty or contains diagnostics information pertaining to some failed SQL-connection.

4.4.3.3 SQL-servers

Each *SQL-server* is a processor, perceived by the SQL-agent to be part of the SQL-implementation, that manages SQL-data.

Each SQL-server:

— manages the SQL-session taking place over the SQL-connection between itself and the SQL-client;

— executes SQL-statements received from the SQL-client, receiving and sending data as required;

— maintains the state of the SQL-session, including the authorization identifier and certain session defaults.

4.4.4 SQL-client modules

An *SQL-client module* is a module that is explicitly created and dropped by implementation-defined (IW003) mechanisms.

An SQL-client module does not necessarily have a name; if it does, the permitted names are implementation-defined (IA094).

An SQL-client module contains zero or more *externally-invoked procedures*. An externally-invoked procedure consists of a single SQL-statement. An externally-invoked procedure is invoked from a compilation unit of a host language.

Exactly one SQL-client module is associated with an SQL-agent at any given time. However, in the case of either direct binding style or SQL/CLI, this may be a default SQL-client module whose existence is not apparent to the user.

4.4.5 User identifiers

A *user identifier* represents a user. The means of creating and destroying user identifiers, and their mapping to real users, is not specified by the ISO/IEC 9075 series.

4.4.6 Roles

A *role* is a potential grantee and grantor of privileges and of other roles. A role can also own schemas and other objects.

A *role authorization* permits a grantee (see Subclause 4.8.11, "Privileges") to use every privilege granted to the role. It also indicates whether the role authorization is grantable, in which case the grantee is authorized to grant the role, to revoke a grant of the role, and to destroy the role.

4.4.7 User mapping concepts

A user mapping pairs an authorization identifier with a foreign server descriptor.

4.4.8 Routine mapping concepts

A routine mapping pairs an SQL-invoked routine with a foreign server descriptor.

NOTE 2 — In this revision of the ISO/IEC 9075 series, the SQL-invoked routine that is paired with a foreign server descriptor is restricted to be an SQL-invoked regular function. It is possible that this restriction will not appear in future editions.

4.4.9 Catalogs and schemas

4.4.9.1 Catalogs

A *catalog* is a named collection of SQL-schemas, foreign server descriptors, and foreign data wrapper descriptors in an SQL-environment. The mechanisms for creating and destroying catalogs are implementation-defined (IW005).

The default catalog name for <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s is initially implementation-defined (ID006) but may be changed by the use of <set catalog statement>s.

4.4.9.2 SQL-schemas

An *SQL-schema*, often referred to simply as a *schema*, is a persistent, named collection of descriptors. An object whose descriptor is in some SQL-schema is known as an *SQL-schema object*.

A schema, the schema objects in it, and the SQL-data described by them are said to be owned by the authorization identifier associated with the schema.

SQL-schemas are created and destroyed by execution of SQL-schema statements (or by implementation-defined (IW006) mechanisms).

4.4.9.3 Information Schema

Every catalog contains an SQL-schema with the name INFORMATION_SCHEMA that includes the descriptors of a number of schema objects, mostly view definitions, that together allow every descriptor in that catalog to be accessed, but not changed, as though it was SQL-data.

The data available through the views in an Information Schema includes the descriptors of the Information Schema itself. It does not include the schema objects or base tables of the Definition Schema (see Subclause 4.4.9.4, "Definition Schema").

Each Information Schema view is so specified that a given user can access only those rows of the view that represent descriptors on which that user has privileges.

4.4.9.4 Definition Schema

The *definition schema* is a fictitious schema with the name DEFINITION_SCHEMA; if it were to exist, the SQL-data in its base tables would describe all the SQL-schemas available in a catalog. The ISO/IEC 9075 series defines it only in order to use it as the basis for the views of the Information Schema.

The structure of the Definition Schema is a representation of the data model of SQL.

4.4.10 Foreign servers and descriptors

A foreign server is a processor that is not part of the SQL-implementation. A foreign server is described by a foreign server descriptor. A foreign server manages data that is not part of the SQL-environment. An SQL-server and an SQL-client can use a foreign server descriptor, which is a catalog element, to communicate with a foreign server. The data managed by a foreign server can be accessed by an SQL-server or an SQL-client through foreign tables, which are SQL-schema elements.

4.4.11 Foreign-data wrappers and descriptors

A foreign-data wrapper provides the mechanism by which an SQL-server can access the data that is managed by a foreign server. A foreign-data wrapper is described by a foreign-data wrapper descriptor.

4.4.12 SQL-data

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

4.5 Tables

A *table* is a collection of zero or more rows where each row is an instance of the row type of the table. In general, rows in a table are unordered; however, rows in a table are ordered if the table is the result of a <query expression> that immediately contains an <order by clause>. The columns of a table are the fields of the row type of the table. Each column has a name and a data type. Each row has exactly one value for each column *C*, and the data type of that value is the data type of *C*.

SQL-data consists entirely of table variables, called *base tables*. An operation that references zero or more base tables and returns a table is called a *query*. The result of a query is called a *derived table*. The rows of a derived table may be ordered.

The rows of a table have a type, called "the row type"; every row of a table has the same row type, which is also the row type of the table. A table that is declared to be based on some structured type is called a "typed table"; its columns correspond in name and declared type to the attributes of the structured type. Typed tables have one additional column, called the "self-referencing column" whose type is a reference type associated with the structured type of the table.

If a typed table *TB1* has an associated structured type *TP1* that is a subtype of some other structured type *TP2*, then *TB1* can be defined to be a "subtable" of a typed table *TB2* whose associated type is *TP2*; *TB2* is, in this case, a "supertable" of *TB1*.

A *view* is a named query, which can be invoked by use of this name. The result of such an invocation is called a *viewed table*.

Some queries, and hence some views, are *updatable*, meaning they can appear as targets of statements that change SQL-data. The results of changes expressed in this way are defined in terms of corresponding changes to base tables.

No two columns of a base table or a viewed table can have the same name. Derived tables, other than viewed tables, may contain more than one column with the same name.

A base table is either a schema object (its descriptor is in a schema; see Subclause 4.8.5, "Base tables and their components") or a module object (its descriptor is in a module; see Subclause 4.11, "Modules"). A base table whose descriptor is in a schema is called a *created base table*, and may be either persistent or temporary (though its descriptor is persistent in either case). A *persistent base table* contains 0 (zero) or more rows of persistent SQL-data. A persistent base table is either a *regular persistent base table* or a *system-versioned table*. System-versioned tables differ from regular persistent base tables in two major respects:

— system-versioned tables have two columns of datetime type whose values are generated automatically;

— the update and delete operations on system-versioned tables maintain those columns and have additional side effects.

A base table declared in a module is always a temporary table, and is called a *declared local temporary table*.

A *temporary table* is an SQL-session object that cannot be accessed from any other SQL-session. A *global temporary table* can be accessed from any associated SQL-client module. A *local temporary table* can be accessed only from the module to which it is local.

A temporary table is empty when an SQL-session is initiated and it is emptied (that is, all its rows are deleted) either when an SQL-transaction is terminated or when an SQL-session is terminated, depending on its descriptor.

4.6 SQL data types

4.6.1 General data type information

Every data value belongs to some data type.

Every data type is either *predefined*, *constructed*, or *user-defined*. Every data type has a name. The name of a predefined or constructed data type is a reserved word specified by that part of the ISO/IEC 9075 series that specifies the data type. The name of a user-defined type is provided in its definition. A user-defined data type is a schema object; see Subclause 4.8.4, "User-defined types".

A predefined data type is a data type specified by the ISO/IEC 9075 series, and is therefore provided by the SQL-implementation. A data type is predefined even though the user is required (or allowed) to provide certain parameters when specifying it (for example, the precision of a number).

A predefined data type is atomic. An atomic type is a data type whose values are not composed of values of other data types. The existence of an operation (SUBSTRING, EXTRACT) that is capable of selecting part of a string or datetime value does not imply that a string or datetime is not atomic.

A constructed type is either atomic or composite. A composite type is a data type each of whose values is composed of zero or more values, each of a declared data type.

4.6.2 Null value

Every data type includes a special value, called the *null value*, sometimes denoted by the keyword NULL. This value differs from other values in the following respects.

— Since the null value is in every data type, the data type of the null value implied by the keyword NULL cannot be inferred; hence NULL can be used to denote the null value only in certain contexts, rather than everywhere that a literal is permitted.

— Although the null value is neither equal to any other value nor not equal to any other value — it is *unknown* whether or not it is equal to any given value — in some contexts, multiple null values are treated together; for example, the <group by clause> treats all null values together.

4.6.3 Predefined types

4.6.3.1 Numeric types

There are three classes of numeric type: *exact numeric*, which includes integer types and types with specified precision and scale; *approximate numeric*, which is essentially binary floating point, and for which a precision may optionally be specified; and *decimal floating point*, which is essentially decimal floating point. A precision may optionally be specified for both approximate numeric and decimal floating point types.

Every number has a *precision* (number of digits), and exact numeric types also have a scale (digits after the radix point). Arithmetic operations may be performed on operands of different or the same numeric type, and the result is of a numeric type that depends only on the numeric type of the operands. If the result cannot be represented exactly in the result type, then whether it is rounded or truncated is implementation-defined (IA002). An exception condition is raised if the result is outside the range of numeric values of the result type, or if the arithmetic operation is not defined for the operands.

4.6.3.2 Character string types

A value of *character string type* is a string (sequence) of characters drawn from some character repertoire. The characters in a character string *S* are all drawn from the same character set *CS*. If *S* is the value of some expression *E*, then *CS* is the character set specified for the declared type of *E*. A character string type is either of fixed length, or of variable length up to some implementation-defined (IL001) maximum. A value of *character large object type* is a string of characters from some character repertoire and is always associated with exactly one character set. A *large object character string* is of variable length, up to some implementation-defined (IL002) maximum that is probably greater than that of other character strings.

A character string may be specified as being based on a specific character set by specifying CHARACTER SET in the data type; a particular character set chosen by the SQL-implementation to be the *national character set* may be specified by specifying NATIONAL CHARACTER, NATIONAL CHARACTER VARYING, or NATIONAL CHARACTER LARGE OBJECT (or one of several syntactic equivalents) as the data type.

4.6.3.3 Binary string types

A value of *binary string type* is a string (sequence) of octets. A binary string type is either of fixed length or of variable length, up to an implementation-defined (IL003) maximum. A value of *binary large object type* is a string of octets. A *binary large object string* is of variable length, up to some implementation-defined (IL004) maximum that is probably greater than that of other binary strings.

4.6.3.4 Boolean type

A value of the *Boolean* data type is either *True* or *False*. The truth value of *Unknown* is sometimes represented by the null value.

4.6.3.5 Datetime types

There are three *datetime types*, each of which specifies values comprising datetime fields.

A value of data type TIMESTAMP comprises values of the datetime fields YEAR (between 0001 and 9999), MONTH, DAY, HOUR, MINUTE and SECOND.

A value of data type TIME comprises values of the datetime fields HOUR, MINUTE and SECOND.

A value of data type DATE comprises values of the datetime fields YEAR (between 0001 and 9999), MONTH, and DAY.

A value of DATE is a valid Gregorian date. A value of TIME is a valid time of day.

TIMESTAMP and TIME may be specified with a number of (decimal) digits of fractional seconds precision.

TIMESTAMP and TIME may also be specified as being WITH TIME ZONE, in which case every value has associated with it a time zone displacement. In comparing values of a data type WITH TIME ZONE, the value of the time zone displacement is disregarded.

4.6.3.6 Interval types

A value of an *interval type* represents the duration of a period of time. There are two classes of intervals. One class, called *year-month intervals*, has an interval precision that includes a YEAR field or a MONTH field, or both. The other class, called *day-time intervals*, has an express or implied interval precision that can include any set of contiguous fields other than YEAR or MONTH.

4.6.3.7 XML type

Values of the XML type are called XML values.

4.6.3.8 JSON type

Values of the JSON type are called SQL/JSON values.

4.6.4 Constructed atomic types: reference types

A *reference type* is a constructed data type, a value of which references (or points to) some site holding a value of the referenced type. The only sites that may be so referenced are the rows of typed tables. It follows that every referenced type is a structured type.

4.6.5 Constructed composite types

4.6.5.1 Collection types

A *collection* comprises zero or more elements of a specified data type known as the *element type*.

An *array* is an ordered collection of not necessarily distinct values, whose elements are referenced by their ordinal position in the array.

An array type is specified by an array type constructor.

An *MD-array* is an ordered collection of not necessarily distinct values, whose elements are referenced by their coordinate in the MD-array.

An MD-array type is specified by an MD-array type constructor.

A multiset is an unordered collection of not necessarily distinct values.

A multiset type is specified by a multiset type constructor.

4.6.5.2 Row types

A row type is a sequence of one or more fields. A value of a row type is the sequence of values of its fields.

4.7 Sites and operations on sites

4.7.1 Sites

A *site* is a place that holds an instance of a value of a specified data type. Every site has a defined degree of persistence, independent of its data type. A site that exists until deliberately destroyed is said to be *persistent*. A site that necessarily ceases to exist on completion of a compound SQL-statement, at the end of an SQL-transaction, or at the end of an SQL-session is said to *temporary*. A site that exists only for as long as necessary to hold an argument or returned value is said to be *transient*.

As indicated above, the principal kind of persistent or temporary site is the base table. A base table is a special kind of site, in that constraints can be specified on its values, which the SQL-implementation is required to enforce (see Subclause 4.8.5.4, "Table constraints").

Some sites may be referenced by their names — for example, base tables and SQL variables (see ISO/IEC 9075-4). Some sites may be referenced by a REF value. A site occupied by an element of an array may be referenced by its element number.

4.7.2 Assignment

The instance at a site can be changed by the operation of *assignment*. Assignment replaces the instance at a site (known as the *target*) with a new instance of a (possibly different) value (known as the *source* value). Assignment has no effect on the reference value of a site, if any.

4.7.3 Nullability

Every site has a *nullability characteristic*, which indicates whether it may contain the null value (is *possibly nullable*) or not (is *known not nullable*). Only the columns of base tables may be constrained to be known not nullable, but columns derived from such columns may inherit the characteristic.

No table can be null, though a table may have no rows.

4.8 SQL-schema objects

4.8.1 General SQL-schema object information

An SQL-schema object has a descriptor. The descriptor of a persistent base table describes a persistent object that has a separate, though dependent, existence as SQL-data. Other descriptors describe SQL objects that have no existence distinct from their descriptors (at least as far as the ISO/IEC 9075 series is concerned). Hence there is no loss of precision if, for example, the term "assertion" is used when "assertion descriptor" would be more strictly correct.

Every schema object has a name that is unique within the schema among objects of the name class to which it belongs. The name classes are:

— base tables and views;

— domains and user-defined types;

— table constraints, domain constraints, and assertions;

— SQL-server modules;

— triggers;

— SQL-invoked routines (specific names only, which are not required to be specified explicitly, but if not are implementation-dependent (UV001));

— character sets;

— collations;

— transliterations;

— sequence generators.

Certain schema objects have named components whose names are required to be unique within the object to which they belong. Thus columns are uniquely named components of base tables or views, attributes are uniquely named components of structured types, and fields are uniquely named components of row types.

Some schema objects may be provided by the SQL-implementation and can neither be created nor dropped by a user.

4.8.2 Descriptors relating to character sets

4.8.2.1 Character sets

A *character set* has a named set of characters (*character repertoire*) that may be used for forming values of the character string type, as well as a named *character encoding form*. Every character set has a *default collation*. Character sets provided by the SQL-implementation, whether defined by other standards or by the SQL-implementation, are represented in the Information Schema.

When characters are contained entirely within an SQL-implementation, the methods for encoding them and for collecting them into strings are implementation-dependent (UW001). When characters are

exchanged with host programs or other entities outside of the SQL-implementation, the character sets also have an encoding, which specifies the bits used to represent each character, and a *character encoding form*, which specifies the scheme used to collect characters into strings.

The character repertoire of every character set supported by an SQL-implementation is some subset of the repertoire of the Universal Character Set specified by ISO/IEC 10646:2020.

The notation specified in ISO/IEC 10646:2020, Subclause 7.5, "Short identifiers for code points (UIDs)", is the canonical representation of characters and character strings in ISO/IEC 9075.

NOTE 3 — ISO/IEC 10646:2020 assigns a range of code points for "private use". Future editions of ISO/IEC 10646:2020 are likely to add more code points, which SQL-implementations are required to support.

4.8.2.2 Collations

A *collation* is a named operation for ordering character strings in a particular character repertoire.

A site declared with a character string type may be specified as having a collation, which is treated as part of its data type.

Every collation shall be defined by or derived from an International Standard such as ISO/IEC 14651 or by a National Standard, or shall be an implementation-defined (IA003) collation.

4.8.2.3 Transliterations

A *transliteration* is a named operation for mapping from a character string of some character set into a character string of a given, not necessarily distinct, character set. The operation is performed by invocation of an external function identified by the name of the transliteration. Since an entire string is passed to this function and a string returned, the mapping is not necessarily from one character to one character, but may be from a sequence of one or more characters to another sequence of one or more characters.

4.8.3 Domains and their components

4.8.3.1 Domains

A *domain* is a named user-defined object that can be specified as an alternative to a data type in certain places where a data type can be specified. A domain consists of a data type, possibly a default option, and zero or more (domain) constraints.

4.8.3.2 Domain constraints

A *domain constraint* applies to every column that is based on that domain, by operating as a table constraint for each such column.

Domain constraints apply only to columns based on the associated domain.

A domain constraint is applied to every value resulting from a cast operation to the domain.

4.8.4 User-defined types

4.8.4.1 Introduction to user-defined types

A user-defined type is a schema object, identified by a <user-defined type name>. A user-defined type is either a *distinct type* or a *structured type*.

4.8.4.2 Distinct types

A *distinct type* is a user-defined data type that is based on some predefined type or collection type. The collection type on which a distinct type is based shall not be an MD-array type. The values of a distinct type are represented by the values of the type on which it is based.

An argument of a distinct type can be passed only to a parameter of the same distinct type. This allows precise control of what routines can be invoked on arguments of that data type.

4.8.4.3 Structured types

A *structured type* is a named, user-defined data type. An *attribute* is a named component of a structured type. Each attribute of a structured type has a data type and a default value.

A value of a structured type comprises a number of *attribute values*. Attribute values are said to be *encapsulated*; that is to say, they are not directly accessible to the user, if at all. An attribute value is accessible only by invoking a function known as an *observer function* that returns that value. A value of a structured type can also be accessed by a *locator*.

A structured type may be defined to be a *subtype* of another structured type, known as its *direct supertype*. A subtype *inherits* every attribute of its direct supertype, and may have additional attributes of its own. An expression of a subtype may appear anywhere that an expression of any of its supertypes is allowed (this concept is known as *substitutability*). Moreover, the value of an expression may be a value of any subtype of the declared type of the expression.

One or more base tables can be created based on a structured type. A base table based on a structured type *ST* can be a *subtable* of a base table based on a supertype of *ST*.

One or more views can be created based on a structured type. A view based on a structured type *ST* can be a subview of a view based on a supertype of *ST*.

4.8.5 Base tables and their components

4.8.5.1 Base tables

A *base table* is a site that holds a table value (see Subclause 4.5, “Tables”). All SQL-data is held in base tables.

If a base table is based on a structured type, it may be a *subtable* of one or more other base tables that are its *supertables*.

4.8.5.2 Columns

A *column* is a field of the row type of the table. It is described by a column descriptor.

4.8.5.3 Periods

A *period* is an object associated with a single base table. A period definition for a given table associates a period name with a pair of column names defined for that table. The columns shall both be of a datetime data type and known not nullable. Further, the declared types of both columns shall be identical.

4.8.5.4 Table constraints

A *table constraint* is an integrity constraint associated with a single base table.

A table constraint is either a *unique constraint*, a *primary key constraint*, a *referential constraint*, or a *check constraint*.

A unique constraint specifies one or more columns of the table as *unique columns*. A unique constraint is satisfied if and only if no two rows in a table have the same non-null values in the unique columns.

A primary key constraint is a unique constraint that specifies PRIMARY KEY. A primary key constraint is satisfied if and only if no two rows in a table have the same non-null values in the unique columns and none of the values in the specified column or columns are the null value.

A referential constraint specifies one or more columns as *referencing columns* and corresponding *referenced columns* in some (not necessarily distinct) base table, referred to as the *referenced table*. Such referenced columns are the unique columns of some unique constraint of the referenced table. A referential constraint is always satisfied if, for every row in the referencing table, the values of the referencing columns are equal to those of the corresponding referenced columns of some row in the referenced table. If null values are present, however, satisfaction of the referential constraint depends on the treatment specified for nulls (known as the *match type*).

Referential actions may be specified to determine what changes are to be made to the referencing table if a change to the referenced table would otherwise cause the referential constraint to be violated.

A table check constraint specifies a *search condition*. The constraint is violated if the result of the search condition is *False* for at least one row of the table (but not if it is *Unknown*).

4.8.5.5 Triggers

A *trigger* is an object associated with a single base table or a single viewed table. A trigger specifies a *subject table*, *trigger event*, a *trigger action time*, and one or more *triggered actions*.

A trigger event specifies what action on the subject table shall cause the triggered actions. A trigger event is either INSERT, DELETE, or UPDATE.

A trigger action time specifies whether the triggered action is to taken BEFORE, INSTEAD OF, or AFTER the trigger event.

A triggered action is either an <SQL procedure statement> or BEGIN ATOMIC, followed by one or more <SQL procedure statement>s terminated with <semicolon>s, followed by END.

4.8.6 View definitions

A *view* (strictly, a *view definition*) is a named query that may for many purposes be used in the same way as a base table. Its value is the result of evaluating the query. See also Subclause 4.5, "Tables".

4.8.7 Assertions

An *assertion* is a check constraint. The constraint is violated if the result of the search condition is *False* (but not if it is *Unknown*).

4.8.8 SQL-server modules (defined in ISO/IEC 9075-4)

An *SQL-server module* is a module that is a schema object. See Subclause 4.11, "Modules".

4.8.9 Schema routines

A *schema routine* is an SQL-invoked routine that is a schema object. See Subclause 4.12, "Routines".

4.8.10 Sequence generators

A *sequence generator* is a mechanism for generating successive exact numeric values, one at a time. A sequence generator is either an *external sequence generator* or an *internal sequence generator*. An external sequence generator is a named schema object while an internal sequence generator is a component of another schema object. A sequence generator has a data type, which shall be an exact numeric type with scale 0 (zero). A sequence generator has a time-varying *current base value*, which is a value of its data type. Refer to Subclause 4.29, "Sequence generators", in ISO/IEC 9075-2 for details.

4.8.11 Privileges

A *privilege* represents a grant, by some grantor, to a specified grantee (which is either an authorization identifier, a role, or PUBLIC), of the authority required to use, or to perform a specified action on, a specified

schema object. The specifiable actions are: SELECT, INSERT, UPDATE, DELETE, REFERENCES, USAGE, UNDER, TRIGGER, and EXECUTE.

A privilege *with grant option* authorizes the grantee to grant that privilege to other grantees, with or without the grant option.

A SELECT privilege *with hierarchy option* automatically provides the grantee with SELECT privileges on all subtables, both existing subtables and all subtables that may be added in the future, of the table on which the privilege is granted.

Every possible grantee is authorized by privileges granted to PUBLIC. SELECT with grant option is granted to PUBLIC for every schema object in the Information Schema.

An authorization identifier who creates a schema object is automatically granted all possible privileges on it, with grant option.

Only an authorization identifier who has some privilege on a schema object is able to discover its existence.

4.9 Integrity constraints and constraint checking

4.9.1 Constraint checking

There are two kinds of schema object that describe constraints: assertions and table constraints (including domain constraints of domains on which columns of that table may be based), and they are checked in the same way.

Table constraints are either *enforced* or *not enforced*. Domain constraints and assertions are always enforced.

Every constraint is either *deferrable* or *not deferrable*.

In every SQL-session, every constraint has a *constraint mode* that is a property of that SQL-session. Each constraint has a (persistent) default constraint mode, with which the constraint starts each SQL-transaction in each SQL-session.

A constraint mode is either *deferred* or *immediate*, and can be set by an SQL-statement, provided the constraint is deferrable.

When a transaction is initiated, the constraint mode of each constraint is set to its default.

On completion of execution of every SQL-statement, every enforced constraint is checked whose constraint mode is immediate.

Before termination of a transaction, every constraint mode is set to immediate (and therefore every enforced constraint is checked).

4.9.2 Determinism and constraints

Expression evaluation results may be non-deterministic. For example, in the case of a column whose data type is varying character string, the value remaining after the elimination of duplicates may be different on different occasions, even though the data is the same. This can occur because the number of trailing whitespace characters may vary from one duplicate to another, and the value to be retained, after the duplicates have been eliminated, is not specified by ISO/IEC 9075. Hence, the length of that value is non-deterministic. In such a case, the expression is said to be *a potential source of non-determinism* ("potential", because it may be that all SQL-agents that ever update that column may remove trailing whitespace; but this cannot be known to the SQL-implementation).

Certain predicates, said to be retrospectively deterministic, may contain CURRENT_DATE, CURRENT_TIMESTAMP, or LOCALTIMESTAMP (potential sources of non-determinism), yet be deterministic because they will remain *True* for all future time.

Because a constraint or assertion that contains a potential source of non-determinism might be satisfied at one time, yet fail at some later time, constraints and assertions must be retrospectively deterministic.

The generation expression of a generated column must not contain a potential source of non-determinism. This insures that the value of a generated column is effectively determined at the last time the row was inserted or updated.

A routine may claim to be deterministic; if it is not, then the effect of invoking the routine is implementation-dependent (UA002).

4.9.3 Consistency when deleting and updating multiple rows

Special considerations are required when certain SQL-statements (particularly <update statement: searched>, <delete statement: searched>, and <merge statement>) cause the modification or deletion of an arbitrary number of rows in SQL tables. Those considerations are reflected in the General Rules of various Subclauses within Clause 15, "Additional data manipulation rules".

Those Subclauses within Clause 15, "Additional data manipulation rules", depend on the concept of other Subclauses (including <update statement: searched>, <delete statement: searched>, and <merge statement>) having, as part of their processing, "marking" identified rows for update processing or deletion processing. When those Subclauses, specifically Subclause 15.8, "Effect of deleting rows from base tables", and Subclause 15.14, "Effect of replacing rows in base tables", are invoked as a result of "applying" their General Rules, they process collections of rows that have been "marked for replacement" or "marked for deletion" by other Subclauses.

Consequently, although Subclause 15.8, "Effect of deleting rows from base tables", and Subclause 15.14, "Effect of replacing rows in base tables", are invoked as though they are "subroutines" (see Subclause 6.3.7, "Subclauses used as subroutines"), they have no explicit parameters. Instead, they operate on those collections of rows that have been previously marked for replacement or deletion.

4.10 Communication between an SQL-agent and an SQL-server

4.10.1 Host languages

A *host language* is a programming language that can be used to write an SQL-agent. For an SQL-implementation to communicate successfully with an SQL-agent, the latter must be written using a host language supported by the SQL-implementation. The set of host languages supported by an SQL-implementation is implementation-defined (IA004). ISO/IEC 9075-2, ISO/IEC 9075-3, and ISO/IEC 9075-10 define bindings between an SQL-implementation and several programming languages, mostly languages for which there is an International Standard. A conforming SQL-implementation is required to support at least one host language for which the ISO/IEC 9075 series defines a binding.

There are several methods of communicating, known as *binding styles*.

— The SQL-client module binding style (specified in ISO/IEC 9075-2). In this binding style, the user, using an implementation-defined (IW007) mechanism, specifies a module to be used as an SQL-client module.

— The Call-Level Interface (specified in ISO/IEC 9075-3). In this case, the SQL-agent invokes one of a number of standard routines, passing appropriate arguments, such as a character string whose value is some SQL-statement.

— Embedded SQL (specified in ISO/IEC 9075-2 and ISO/IEC 9075-10). In this case, SQL-statements are coded into the application program; an implementation-dependent (UW002) mechanism is then used to do the following.

  • Generate from each SQL-statement an externally-invoked procedure. These procedures are collected together into a module, for subsequent use as an SQL-client module. In Object Lan-

guage Bindings, described in ISO/IEC 9075-10, the roles of modules and procedures are served by classes and their methods.

- Replace each SQL-statement with an invocation of the externally-invoked procedure or method generated from it.

— Direct invocation of SQL (specified in ISO/IEC 9075-2). Direct invocation is a method of executing SQL-statements directly, through a front-end that communicates directly with the user.

ISO/IEC 9075-2 specifies the actions of an externally-invoked procedure in an SQL-client module when it is called by a host language program. ISO/IEC 9075-10 specifies the actions of a method of an SQL-client class when it is invoked from a host program.

4.10.2 Source language character set

The text of SQL language constructs is represented by characters as defined by ISO/IEC 10646:2020. Except for <identifier>s and <character string literal>s, the specific characters are well-identified by ISO/IEC 9075 — they are all characters that appear in ISO/IEC 646 (although not every character in ISO/IEC 646 is currently used in the representation of the SQL language) in the ranges of U+0021 through U+007E. In addition, representation of SQL text requires the use of *whitespace*; minimally, U+0020, Space, is required, and it is implementation-defined (IA005) whether (and which) other Unicode characters with the Unicode property White_Space are permitted to be used where ISO/IEC 9075 specifies the use of whitespace.

In <character string literal>s, ISO/IEC 9075 permits the use of every Unicode character, subject to implementation-defined (IA006) restrictions and further restricted by the rules of the character set specified by a <character set specification> immediately contained in the <character string literal>. <character string literal>s should not contain characters equivalent to U+0008, Backspace; this recommendation does not apply to character string values stored in sites.

<identifier>s shall not incorporate characters that are equivalent to Unicode characters with Unicode categories "Cc", "Cf", "Cn", "Cs", "Zl", or "Zp". <identifier>s should not incorporate characters that are equivalent to Unicode characters with Unicode category "Co"[3]. <identifier>s should not incorporate characters that are equivalent to Unicode characters with Unicode category "Zs" other than U+0020, Space.

NOTE 4 — The Unicode General Category classes "Cc", "Cf", "Cn", "Cs", "Zl", and "Zp" are assigned to Unicode characters that are, respectively, Control codes (other), Formal Control codes, Not Assigned codes, Surrogates, Line Separators, and Paragraph Separators. Unicode General Category class "Co" identifies private use characters. Unicode General Category class "Zs" identifies Space Separators (whitespace).

4.10.3 Parameter passing and data type correspondences

4.10.3.1 General parameter passing and data type correspondence information

Each parameter in the parameter list of an externally-invoked procedure has a name and a data type. The method and time of binding between an external routine's reference to a compilation unit and that compilation unit is implementation-defined (IW008).

4.10.3.2 Data type correspondences

ISO/IEC 9075 specifies correspondences between SQL data types and host language data types. Not every SQL data type has a corresponding data type in every host language.

3 The use of characters with Unicode General Category "Co" is likely to have serious effects on the portability of SQL code.

4.10.3.3 Locators

A host variable, host parameter, SQL parameter of an external routine, or the value returned by an external function may be specified to be a *locator*. The purpose of a locator is to allow very large data instances to be operated upon without transferring their entire value to and from the SQL-agent. Refer to Subclause 4.37.5, "Locators", in ISO/IEC 9075-2 for details.

4.10.3.4 Status parameters

Every externally-invoked procedure is required to have an output parameter called SQLSTATE, which is known as a *status parameter*.

SQLSTATE is a character string of length 5, whose values are defined by the parts of the ISO/IEC 9075 series. An SQLSTATE value of '00000' (five zeros) indicates that the most recent invocation of an externally-invoked procedure was successful.

4.10.3.5 Indicator parameters

An *indicator parameter* is an integer parameter that, by being specified immediately following a parameter (other than an indicator parameter), is associated with it. A negative value in an indicator parameter indicates that the associated parameter is null. A value greater than zero indicates what the length of the value of the associated parameter would have been, had it not been necessary to truncate it. This may arise with a character string and with certain other data types.

If a null value is to be assigned to a parameter that has no associated indicator parameter, then an exception condition is raised.

4.10.4 Descriptor areas

A *descriptor area* (not to be confused with a descriptor) is a named area allocated by the SQL-implementation at the request of the SQL-agent. A descriptor area is used for communication between the SQL-implementation and the SQL-agent. There are SQL-statements for transferring information between the SQL-agent and a descriptor area.

4.10.5 Diagnostic information

A *diagnostics area* is a communication area allocated by the SQL-implementation in which one or more *conditions* are recorded as they arise.

Whenever the SQL-implementation executes an SQL-statement (other than an SQL-diagnostics statement), it sets values, representing one or more conditions resulting from that execution, in a diagnostics area. These values give some indication of what has happened. They can be accessed by SQL-diagnostics statements. Execution of an SQL-diagnostics statement does not cause any conditions to be recorded.

A conforming SQL-implementation is not required to set more than one condition at the same time.

Multiple diagnostics areas can exist when an SQL-statement *SS* is being executed by an SQL-server. This happens in certain specific circumstances when execution of *SS* causes other SQL-statements to be executed, synchronously, before the execution of *SS* is complete. The multiple diagnostics areas form a pushdown stack, the first element of which contains information pertaining to the most recently executed SQL-statement. Two examples of when additional diagnostics areas come into existence are when *SS* is or contains a <call statement> and when execution of *SS* causes a triggered action to be executed.

4.10.6 SQL-transactions

An *SQL-transaction* (*transaction*) is a sequence of executions of SQL-statements that is atomic with respect to recovery. That is to say: either the execution result is completely successful, or it has no effect on any SQL-schemas or SQL-data.

At any time, there is at most one current SQL-transaction between the SQL-agent and the SQL-implementation.

If there is no current SQL-transaction, execution of a *transaction-initiating statement* or evaluation of a <subquery> will initiate one.

Every SQL-transaction is terminated by either a commit statement or a rollback statement. The execution of either of these statements may be implicit.

An SQL-transaction has a *transaction state*. Certain properties of the transaction state are set by the execution of SQL-statements. Such SQL-statements may be executed only when there is no SQL-transaction current. On the first occasion, at or after a transaction is initiated, that the SQL-client connects to, or sets the connection to, an SQL-server, the properties are sent to that SQL-server.

The *transaction access mode* of an SQL-transaction indicates whether the transaction is read-only (is not permitted to change any persistent SQL-data) or read-write (is permitted to change persistent SQL-data).

The *transaction isolation level* of an SQL-transaction specifies the extent to which the effects of actions by SQL-agents in the SQL-environment are perceived within that SQL-transaction.

Every isolation level guarantees that every SQL-transaction is executed; SQL-transactions not executing completely fail completely. Every isolation level guarantees that no update is lost. The highest isolation level, SERIALIZABLE, guarantees *serializable* execution, meaning that the effect of SQL-transactions that overlap in time is the same as the effect they would have had, had they not overlapped in time. The other levels of isolation, REPEATABLE READ, READ COMMITTED, and READ UNCOMMITTED, guarantee progressively lower degrees of isolation.

4.11 Modules

There are two kinds of modules, each of which has certain properties and contains various kinds of module objects (also known as module contents). The principal module objects are one or more routines (see Subclause 4.12, "Routines").

A module is one of the following:

— an SQL-client module, containing only externally-invoked procedures;

— an SQL-server module, containing only SQL-invoked routines.

4.12 Routines

4.12.1 General routine information

Table 1, "Relationships between externally-invoked and SQL-invoked routines", shows the terms used for the various possible combinations of routine written in SQL and written in another language, as well as routines invoked from SQL and routines invoked from another language.

Table 1 — Relationships between externally-invoked and SQL-invoked routines

	Routines written in SQL	Routines written in languages other than SQL
Routines invoked from SQL	SQL functions and SQL procedures	External functions and procedures
Routines invoked from languages other than SQL	Externally-invoked procedures	*(not relevant to SQL)*

An *SQL-invoked routine* is a routine that can be invoked from SQL. It is either a function or a procedure. Some functions have special properties that characterize them as *methods*.

An SQL-invoked routine is either a schema object or a component of an SQL-server module (itself a schema object).

An *SQL-invoked procedure* is a procedure invoked by an SQL call statement. An *SQL-invoked function* is invoked by a routine invocation in some value expression.

An *external routine* is an SQL-invoked routine that references some compilation unit of a specified programming language that is outside the SQL-environment. The mechanism and time of binding of such a reference is implementation-defined (IW008).

An *SQL routine* is an SQL-invoked routine whose routine body is written in SQL.

The name of an SQL-invoked routine is not required to be unique. If two or more routines share the same name, that name is said to be *overloaded*, and an invocation of that name will cause execution of the routine whose signature best matches the arguments of the invocation. Normally, only the declared types of the expressions denoting the argument values are considered in determining the best match, but in the case of methods, which are invoked using a distinguishing syntax, the most specific type of one of the arguments is taken into consideration.

4.12.2 Type preserving functions

If an SQL-invoked function has a parameter that is specified RESULT, that parameter is known as a *result parameter*, and if the data type of the result parameter and that of the result are the same user-defined type, then the function is said to be a *type preserving function*. The result of a type preserving function is the value of the result parameter, possibly mutated.

Every mutator function is a type preserving function.

4.13 SQL-statements

4.13.1 Classes of SQL-statements

An SQL-statement is a string of characters that conforms to the Format and Syntax Rules specified in one of the parts of the ISO/IEC 9075 series.

Most SQL-statements can be prepared for execution and executed in an SQL-client module. In this case, an externally-invoked procedure with a single SQL-statement is created when the SQL-statement is prepared and that externally-invoked procedure is implicitly called whenever the prepared SQL-statement is executed.

There are at least five ways of classifying SQL-statements.

— According to their effect on SQL objects, whether persistent objects (i.e., SQL-data, SQL-schemas and their contents, or SQL-client modules) or temporary objects (such as SQL-sessions and other SQL-statements).

— According to whether or not they initiate an SQL-transaction, or can, or shall, be executed when no SQL-transaction is active.

— According to whether or not they may be embedded in a program written in a host language.

— According to whether or not they may be directly executed.

— According to whether or not they may be dynamically prepared and executed.

ISO/IEC 9075 permits SQL-implementations to provide additional, implementation-defined (IE005), statements that may fall into any of these categories.

4.13.2 SQL-statements classified by function

The following are the broad classes of SQL-statements.

— SQL-schema statements that can be used to create, alter, and drop schemas and schema objects.

— SQL-data statements that perform queries, and insert, update, and delete operations on tables. Execution of an SQL-data statement is capable of affecting more than one row, of more than one table.

— SQL-transaction statements that set parameters for, and start or terminate transactions.

— SQL-control statements that may be used to control the execution of a sequence of SQL-statements.

— SQL-connection statements that initiate and terminate connections, and allow an SQL-client to switch from an SQL-session with one SQL-server to an SQL-session with another.

— SQL-session statements that set some default values and other parameters of an SQL-session.

— SQL-diagnostics statements that get diagnostics (from the diagnostics area) and signal exceptions in SQL routines.

— SQL-dynamic statements that support the preparation and execution of dynamically-generated SQL-statements, and obtaining information about them.

5 Parts of the ISO/IEC 9075 series

5.1 Overview

This document, Framework, is a prerequisite for all other parts, because it describes the basic concepts on which other parts are based and the notation used in them.

ISO/IEC 9075-2, Foundation, specifies the structure of SQL-statements and the effects of executing them.

Every part of the ISO/IEC 9075 series other than parts 1 and 2 is specified as an amendment to other parts of the ISO/IEC 9075 series. See Subclause 6.3.6, "Relationships of parts within the ISO/IEC 9075 series", for details.

ISO/IEC 9075-3, Call-Level Interface, specifies another mechanism of communication between an SQL-agent and an SQL-implementation.

ISO/IEC 9075-4, Persistent Stored Modules, specifies significant additions to SQL itself by making SQL computationally complete.

ISO/IEC 9075-9, Management of External Data, specifies significant additions to SQL that permit an SQL-agent to access data not under the control of SQL-servers in the SQL-environment.

ISO/IEC 9075-10, Object Language Bindings, specifies the manner in which SQL statements can be embedded into Java programs.

ISO/IEC 9075-11, Information and Definition Schemas, specifies views by which an application may learn the names of persistent database objects, such as tables, views, columns, etc.

ISO/IEC 9075-13, SQL Routines and Types using the Java™ Programming Language, specifies the ability to invoke static methods written in the Java programming language as SQL-invoked routines and to use classes defined in the Java programming language as SQL structured types.

ISO/IEC 9075-14, XML-Related Specifications, defines ways in which Database Language SQL can be used in conjunction with XML.

ISO/IEC 9075-15, Multidimensional Arrays, defines ways in which Database Language SQL can be used with multidimensional array data.

ISO/IEC 9075-16, Property Graph Queries, defines ways that allow Database Language SQL to represent property graphs and interact with them.

The content of each part is described in the following subclauses.

5.2 ISO/IEC 9075-1: Framework (SQL/Framework)

This document contains the following material.

— A description of an SQL-environment, and brief descriptions of the concepts used in the ISO/IEC 9075 series.

— A brief description of the content of each part. These descriptions are purely informative and do not constitute requirements.

— Notations and conventions that apply to all or most parts of the ISO/IEC 9075 series. Other parts specify further conventions as required.

5.3 ISO/IEC 9075-2 Foundation (SQL/Foundation)

5.3.1 Introduction to ISO/IEC 9075-2 Foundation (SQL/Foundation)

ISO/IEC 9075-2 SQL/Foundation, specifies concepts, syntax, and semantics for operations on SQL-data via SQL-statements. SQL/Foundation includes specifications for:

— Data types, such as string and numeric types;

— Schema objects, such as tables and views;

— Scalar expressions, such as numeric and datetime expressions;

— Query expressions which yield table values, and various kinds of table references;

— Predicates, such as comparison predicates and row-pattern (match) predicates;

— Aggregate functions, such as set functions, that compute values from a group of rows.

5.3.2 Bindings methods

5.3.2.1 Introduction to bindings methods

ISO/IEC 9075-2 specifies three methods of binding an SQL-agent to an SQL-implementation.

5.3.2.2 Embedded SQL

Embedded SQL is a method of embedding SQL-statements in a compilation unit that is otherwise written according to the rules for a particular programming language, known as the *host language*. It defines how an equivalent compilation unit, entirely in the host language, may be derived. In that equivalent compilation unit, each embedded SQL-statement has been replaced by one or more statements that invoke a database language procedure that contains the SQL-statement.

5.3.2.3 Dynamic SQL

Dynamic SQL is an extension of Embedded SQL. Facilities are specified to perform the following.

— Allocate and free a descriptor area used for communication between the SQL-implementation and the SQL-agent.

— Cause the execution of SQL-statements, including the preparation of statements for subsequent execution.

5.3.2.4 Direct invocation of SQL

Direct invocation of SQL is a method of executing SQL-statements directly. The specific methods involved in direct invocation of SQL are implementation-defined (IW009).

5.3.3 SQL-statements specified in ISO/IEC 9075-2

The following are the classes of SQL-statements specified in ISO/IEC 9075-2.

— SQL-schema statements, which can be used to create, alter, and drop schemas and the schema objects specified in ISO/IEC 9075-2.

— SQL-data statements, which can be used to perform queries, and insert, update and delete operations on tables. Some SQL-data statements contain the word “dynamic” in their names; they are not to be confused with SQL-dynamic statements.

— SQL-transaction statements, which can be used to set properties of, and initiate or terminate transactions.

— SQL-control statements.

— SQL-connection statements, which can be used to initiate and terminate connections, and allow an SQL-client to switch from an SQL-session with one SQL-server to an SQL-session with another.

— SQL-session statements, which can be used to set some default values and other properties of an SQL-session.

— SQL-diagnostics statements, which get diagnostic information (from the diagnostics area).

— SQL-dynamic statements, which support the preparation and execution of dynamically generated SQL-statements, and obtaining information about them.

— SQL embedded exception declaration, which is converted to a statement in the host language.

For each SQL-statement that it defines, ISO/IEC 9075-2 specifies which SQL-statements will, if executed when no transaction is active, initiate a transaction and which will not.

For each SQL-statement, ISO/IEC 9075-2 specifies whether:

— it may be embedded in a host language;

— it may be dynamically prepared and executed;

NOTE 5 — Preparable SQL-statements can be executed immediately, with the exception of those that fetch data into a descriptor area.

— it may be executed directly.

5.4 ISO/IEC 9075-3: Call-Level Interface (SQL/CLI)

ISO/IEC 9075-3 specifies a method of binding between a program, written in one of a number of programming languages, and an SQL-implementation. The effect is functionally equivalent to dynamic SQL, specified in ISO/IEC 9075-2.

Procedures (routines) are specified that can be used to:

— allocate and free resources (descriptor, or communication areas);

— initiate, control, and terminate SQL-connections between SQL-client and SQL-servers;

— cause the execution of SQL-statements, including the preparation of statements for subsequent execution;

— obtain diagnostic information;

— obtain information about the SQL-implementation, for example, the SQL-servers to which the SQL-client may be able to connect.

An important difference between CLI and Bindings is that, in the context of the latter, there is only one SQL-environment, whereas, in the context of CLI, a number of SQL-environments can be initiated and managed independently. Consequently, although an SQL-environment is defined to be simply a set of circumstances with various features, the term is used in CLI to refer to the current state (descriptor) of one SQL-environment, possibly among many. Thus, the term is used to mean the session between an application (SQL-agent) and an SQL-client (not to be confused with the SQL-session — referred to in CLI as the SQL-connection — between SQL-client and SQL-server).

5.5 ISO/IEC 9075-4: Persistent Stored Modules (SQL/PSM)

5.5.1 Introduction to SQL/PSM

ISO/IEC 9075-4 makes SQL computationally complete by specifying the syntax and semantics of additional SQL-statements.

Those include facilities for the following actions.

— The specification of statements to direct the flow of control.

— The assignment of the result of expressions to variables and parameters.

— The specification of condition handlers that allow compound statements to deal with various conditions that may arise during their execution.

— The specification of statements to signal and resignal conditions.

— The declaration of standing SQL-server cursors.

— The declaration of local variables.

It also defines Information Schema tables that contain schema information describing SQL-server modules.

5.5.2 SQL-statements specified in ISO/IEC 9075-4

The following are the broad classes of SQL-statements specified in ISO/IEC 9075-4.

— Additional SQL-control statements, which may be used to control the execution of an SQL routine.

— SQL-control declarations, which may be used to declare variables and exception handlers.

— Additional SQL-diagnostics statements, which may be used to signal exceptions.

— Additional SQL-schema statements, which may be used to create and drop modules.

5.6 ISO/IEC 9075-9: Management of External Data (SQL/MED)

ISO/IEC 9075-9 defines facilities that allow Database Language SQL to support management of external data through the use of foreign tables and datalink data types.

These include facilities for defining the following objects:

— foreign servers;

— foreign-data wrappers;

— foreign tables.

There are also facilities for defining routines that can be used by an SQL-server to communicate with a foreign-data wrapper to interact with an external data source.

5.7 ISO/IEC 9075-10: Object Language Bindings (SQL/OLB)

ISO/IEC 9075-10 defines facilities for the embedding of SQL statements in Java programs.

5.8 ISO/IEC 9075-11: Information and Definition Schemas (SQL/Schemata)

ISO/IEC 9075-11, Information and Definition Schemas, specifies two schemas, the Information Schema (INFORMATION_SCHEMA) and the Definition Schema (DEFINITION_SCHEMA). The views of the Information Schema enable an application to learn the names of persistent database objects, such as

tables, views, columns, and so forth. These views are defined in terms of the base tables of the Definition Schema. The only purpose of the Definition Schema is to provide a data model to support the Information Schema and to assist understanding. An SQL-implementation need do no more than simulate the existence of the Definition Schema, as viewed through the Information Schema views.

5.9 ISO/IEC 9075-13: SQL Routines and Types Using the Java™ Programming Language (SQL/JRT)

ISO/IEC 9075-13 defines facilities that allow Database Language SQL to enable invocations of static methods written in the Java programming language as SQL-invoked routines, and to use classes defined in the Java programming language as SQL structured types.

These include:

— extensions to the definition, manipulation, and invocation of SQL-invoked routines;

— extensions to the definition and manipulation of user-defined types;

— new built-in procedures.

5.10 ISO/IEC 9075-14: XML-Related Specifications (SQL/XML)

ISO/IEC 9075-14 defines facilities that allow Database Language SQL to enable creation and manipulation of XML documents.

These include:

— a new predefined type, XML;

— new built-in operators to create and manipulate values of the XML type;

— rules for mapping tables, schemas, and catalogs to XML documents.

5.11 ISO/IEC 9075-15: Multidimensional Arrays (SQL/MDA)

ISO/IEC 9075-15 defines facilities that allow Database Language SQL to support creation and manipulation of multidimensional arrays.

These include:

— a new collection type, MD-array;

— new built-in operators to create and manipulate values of the MD-array type.

5.12 ISO/IEC 9075-16: Property Graph Queries (SQL/PGQ)

ISO/IEC 9075-16 defines facilities that allow Database Language SQL to represent property graphs and to interact with them.

These facilities include:

— syntax and semantics to define property graphs (called “SQL-property graphs”) as components of SQL-schemas;

— descriptors of SQL-property graphs, each of which includes a descriptor of a “pure” property graph and a descriptor of a “tabular” property graph; property graphs comprise sets of vertices and edges that are described in the various descriptors;

— a new alternative to SQL’s <table primary> production that allows reference to a <graph table>;

— new syntax and semantics that support matching patterns in property graphs.

6 Notation and conventions used in other parts of the ISO/IEC 9075 series

6.1 Notation taken from ISO/IEC 10646:2020

The notation for the representation of UCS code points is defined in ISO/IEC 10646:2020, Subclause 7.5, "Short identifiers for code points".

In this document, this notation is used only to unambiguously identify characters and is not meant to imply a specific encoding for an SQL-implementation's use of that character.

6.2 Notation provided in the ISO/IEC 9075 series

The syntactic notation used in the ISO/IEC 9075 series is an extended version of BNF ("Backus Normal Form" or "Backus Naur Form").

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

In the version of BNF used in the ISO/IEC 9075 series, the symbols have the meanings shown in Table 2, "Symbols used in BNF".

Table 2 — Symbols used in BNF

Symbol	Meaning
< >	A character string enclosed in angle brackets is the name of a syntactic element (BNF non-terminal) of the SQL language.
: : =	The definition operator is used in a production rule to separate the element defined by the rule from its definition. The element being defined appears to the left of the operator and the formula that defines the element appears to the right.
[ ]	Square brackets indicate optional elements in a formula. The portion of the formula within the brackets may be explicitly specified or may be omitted.
{ }	Braces group elements in a formula. The portion of the formula within the braces shall be explicitly specified.
|	The alternative operator. The vertical bar indicates that the portion of the formula following the bar is an alternative to the portion preceding the bar. If the vertical bar appears at a position where it is not enclosed in braces or square brackets, it specifies a complete alternative for the element defined by the production rule. If the vertical bar appears in a portion of a formula enclosed in braces or square brackets, it specifies alternatives for the contents of the innermost pair of such braces or brackets.

Symbol	Meaning
. . .	The ellipsis indicates that the element to which it applies in a formula may be repeated an arbitrary number of times. If the ellipsis appears immediately after a closing brace "}", then it applies to the portion of the formula enclosed between that closing brace and the corresponding opening brace "{". If an ellipsis appears after any other element, then it applies only to that element. In Syntax Rules, Access Rules, General Rules, and Conformance Rules, a reference to the *n*-th element in such a list assumes the order in which these are specified, unless otherwise stated.
! !	Introduces either a reference to the Syntax Rules, used when the definition of a syntactic element is not expressed in BNF, or the Unicode code point or code point sequence that define the character(s) of the BNF production.

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

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

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

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

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

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

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

5) Apply steps Step 1) through Step 5) to each BNF non-terminal symbol that remains in the string.

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

The left normal form derivation of a character string *CS* in the source language character set from a BNF non-terminal *NT* is obtained by applying Step 1) through Step 5) above to *NT*, always selecting in Step 5) the leftmost BNF non-terminal.

6.3 Conventions

6.3.1 Specification of syntactic elements

Syntactic elements are specified in terms of several components.

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

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

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

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

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

— **Conformance Rules**: A specification of how the element shall be supported for conformance to SQL. Conformance Rules are effectively a kind of Syntax Rule, differentiated only because of their use to specify conformance to the SQL language. Conformance Rules in a given Subclause are therefore always applied before the Syntax Rules of that Subclause, and are not normally applied to the result of syntactic transformations. However, in a few cases, Conformance Rules are explicitly applied to syntactic transformations as defined in the Syntax Rules.

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

6.3.2 Specification of the Information and Definition Schemata

The objects of the Information and Definition Schemata in the ISO/IEC 9075 series are specified in terms of several concepts.

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

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

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

— **Initial Table Population**: A specification of the rows of the base table that an SQL-implementation must provide that are not specified in the General Rules of other Subclauses.

— **Conformance Rules**: A specification of how the element shall be supported for conformance to SQL.

The only purpose of the view definitions in the Information Schema is to specify the contents of those viewed tables. The actual objects on which these views are based are implementation-dependent (UV002).

6.3.3 Use of terms

6.3.3.1 Syntactic containment

Let <A>, <B>, and <C> be syntactic elements; let *A1*, *B1*, and *C1* respectively be instances of <A>, <B>, and <C>.

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

In SQL language, *A1* is said to *immediately contain B1* if <A> immediately contains <B> and *B1* is part of the text of *A1*. *A1* is said to *contain* or *specify C1* if *A1* immediately contains *C1* or if *A1* immediately contains *B1* and *B1* contains *C1*. If *A1* contains *C1*, then *C1* is *contained in A1* and *C1* is *specified by A1*.

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

A1 simply contains B1 if *A1* contains *B1* without an intervening instance of <A> or an intervening instance of <B>. If *A1 simply contains B1*, then *B1* is *simply contained in A1*.

A1 directly contains B1 if *A1* contains *B1* without an intervening instance of <subquery>, <within group specification>, or <set function specification> that is not an <ordered set function>.

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

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

Let *VN* be a <table name> that identifies a view. The <query expression> *referenced by VN* is

Case:

— If *VN* is simply contained in an <only spec>, a <target table> that specifies ONLY, or an <insertion target>, then the original <query expression> of *V*.

— Otherwise, the hierarchical <query expression> of *V*.

If <A> contains a <table name> *VN* that identifies a view, then <A> is said to *generally contain* the <query expression> referenced by *VN*. If <A> contains a <query name> that identifies a <query expression> *QE*, then <A> is said to *generally contain QE*. If <A> is a <column reference> that is a column reference, there is an implicit or explicit range variable *RV* qualifying that column reference. If *RV* is a <table name> *VN* that identifies a view, then <A> is said to *generally contain* the <query expression> referenced by *VN*. If *RV* is a <query name> that identifies a <query expression> *QE*, then <A> is said to *generally contain QE*. If <A> contains a <property graph name> *PGN*, then <A> is said to generally contain each <query expression>, if any, and each <table name> that identifies a view, if any, included in the tabular property graph descriptor included in the SQL-property graph descriptor describing the SQL-property graph identified by *PGN*. If <A> contains <B>, then <A> generally contains <B>. If <A> generally contains <B> and <B> generally contains <C>, then <A> generally contains <C>.

Let *RI* be a <routine invocation>, <method invocation>, <static method invocation>, or <method reference>. If <A> contains *RI*, then <A> is said to *broadly contain* the <SQL routine body> of the SQL routine, if any, that is the subject routine of *RI*. If <A> generally contains <B>, then <A> broadly contains <B>. If <A> broadly contains <B> and <B> broadly contains <C>, then <A> broadly contains <C>.

NOTE 6 — The "subject routine of a <routine invocation>" is determined in Subclause 9.18, "Invoking an SQL-invoked routine", in ISO/IEC 9075-2.

NOTE 7 — Broad containment is only used to specify optional features, such as implementation-defined elements or conformance Features.

In a Format, the verb "to be" (including all its grammatical variants, such as "is") is defined as follows: <A> is said to be <B> if there exists a BNF production rule of the form <A> ::= <B>. If <A> is <B> and <B> is <C>, then <A> is <C>. If <A> is <C>, then <C> is said to *constitute* <A>. In SQL language, *A1* is said to be *B1* if <A> is <B> and the text of *A1* is the text of *B1*. Conversely, *B1* is said to *constitute A1* if *A1* is *B1*.

6.3.3.2 Terms denoting rule requirements

In the Syntax Rules, the term *shall* defines conditions that are required to be true of syntactically conforming SQL language. When such conditions depend on the contents of one or more schemas, they are required to be satisfied just before the actions specified by the General Rules are performed. The treatment of language that does not conform to the SQL Formats and Syntax Rules is implementation-dependent (UA004). If one or more conditions required by Syntax Rules are not satisfied when the evaluation of Access or General Rules is attempted and the SQL-implementation is neither processing non-conforming

SQL language nor processing conforming SQL language in a non-conforming manner, then an exception condition is raised: *syntax error or access rule violation (42000)*.

In the Access Rules, the term *shall* defines conditions that are required to be satisfied for the successful application of the General Rules. If any such condition is not satisfied when the General Rules are applied, then an exception condition is raised: *syntax error or access rule violation (42000)*.

In the Conformance Rules, the term *shall* defines conditions that are required to be satisfied if the named Feature is or Features are not supported.

6.3.3.3 Rule evaluation order

A conforming SQL-implementation is not required to perform the exact sequence of actions defined in the General Rules, provided its effect on SQL-data and schemas, on host parameters and host variable, and on SQL parameters and SQL variables is identical to the effect of that sequence. The term *effectively* is used to emphasize actions whose effect might be achieved in other ways by an SQL-implementation.

The Access Rules and Conformance Rules for contained syntactic elements are effectively applied at the same time as the Access Rules and Conformance Rules for the containing syntactic elements. Similarly the Syntax Rules for contained syntactic elements are effectively applied at the same time as the Syntax Rules for the containing syntactic elements. The General Rules for contained syntactic elements are effectively applied before the General Rules for the containing syntactic elements.

Where the precedence of operators is determined by the Formats of the ISO/IEC 9075 series or by parentheses, those operators are effectively applied in the order specified by that precedence.

Where the precedence is not determined by the Formats or by parentheses, effective evaluation of expressions is *generally* performed from left to right. However, it is implementation-dependent (US001) whether expressions are *actually* evaluated left to right, particularly when operands or operators might cause conditions to be raised or if the results of the expressions can be determined without completely evaluating all parts of the expression.

If some syntactic element contains more than one other syntactic element, then the General Rules for contained elements that appear earlier in the production for the containing syntactic element are applied before the General Rules for contained elements that appear later.

For example, in the production:

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

<B> ::=
  ...

<C> ::=
  ...
```

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

There are exceptions to this practice. These are identified either using the phrase “the General Rules of X are evaluated”, or having the form “the General Rules of X, are applied with ...” where X identifies a syntactic element occurring in the Format of the Subclause. Where these phrases occur, the General Rules for the syntactic element identified by X are not evaluated at the normal time but are deferred. For example, the General Rules of Subclause 9.17, “Executing an <SQL procedure statement>”, in ISO/IEC 9075-2, use the phrase “The General Rules of *S* are evaluated” to specify that (effectively), the evaluation of the General Rules of Subclause 9.17, “Executing an <SQL procedure statement>”, is interrupted tem-

porarily to evaluate the General Rules associated with the SQL statement (e.g., an <insert statement>) represented by the symbol *S*.

If the result of an expression or search condition is not dependent on the result of some part of that expression or search condition, then that part of the expression or search condition is said to be *inessential*. An invocation of an SQL-invoked function that is deterministic and that does not possibly modify SQL-data is considered to be *inessential*. It is implementation-defined (IA008) whether an invocation of an SQL-invoked function that is non-deterministic or possibly modifies SQL-data is considered to be *essential* or *inessential*.

If an Access Rule pertaining to an inessential part is not satisfied, then the *syntax error or access rule violation (42000)* exception condition is raised regardless of whether or not the inessential parts are actually evaluated. If evaluation of an inessential part would cause an exception condition to be raised, then it is implementation-dependent (UA005) whether or not that exception condition is raised.

During the computation of the result of an expression, the SQL-implementation may produce one or more *intermediate results* that are used in determining that result. The declared type of a site that contains an intermediate result is implementation-dependent (UV003).

In various parts of the ISO/IEC 9075 series, reference is occasionally made in one Subclause to the Syntax Rules of some other Subclause. For example, in ISO/IEC 9075-2, Subclause 4.48.3, "SQL/JSON data model", there is a paragraph that states "Two SQL/JSON items are *equality-comparable* if one of them is the SQL/JSON null, or if both are SQL/JSON scalars whose SQL types are comparable and acceptable as operands of an equality operation according to the Syntax Rules of Subclause 9.11, "Equality operations"."

The phrase "according to the Syntax Rules..." means that the Syntax Rules, Access Rules, and Conformance Rules of the referenced Subclause shall all be satisfied in order for the specified condition to be satisfied.

6.3.3.4 Conditional rules

A conditional rule is specified with "If" or "Case" conventions. A rule specified with "Case" conventions includes a list of conditional subrules using "If" conventions. The first such "If" subrule whose condition is satisfied is the effective subrule of the "Case" rule. The last subrule of a "Case" rule may specify "Otherwise", in which case it is the effective subrule of the "Case" rule if no preceding "If" subrule in the "Case" rule is satisfied. If the last subrule does not specify "Otherwise", and if there is no subrule whose condition is satisfied, then there is no effective subrule of the "Case" rule.

6.3.3.5 Syntactic substitution

In the Syntax and General Rules, the phrase " *X* is implicit" indicates that the Syntax and General Rules are to be interpreted as if the element *X* had actually been specified. Within the Syntax Rules of a given Subclause, it is known whether the element was explicitly specified or is implicit.

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

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

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

Case:

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

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

6.3.3.6 Other terms

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

An SQL-statement *S1* is said to be executed as a *direct result* of the execution an SQL-statement *S2* if *S2* is a <call statement> *CS* and *S1* is the outermost SQL-statement contained in the SQL-invoked routine that is the subject routine of the <routine invocation> contained in *CS*.

An SQL-statement *S1* is said to be executed as an *indirect result* of executing an SQL-statement *S2* if *S1* is executed in a trigger execution context that has been activated by the execution of *S2*.

A value *P is part of* a value *W* if and only if exactly one of the following is true.

— *W* is a table and *P* is a row of *W*.

— *W* is a row and *P* is a field of *W*.

— *W* is a collection and *P* is an element of *W*.

— *P* is a part of some value that is a part of *W*.

If a value has parts, then it follows that an instance of that value has parts; hence the site it occupies has parts, each of which is also a site.

An item *X is part of* an item *Y* if and only if exactly one of the following is true.

— *Y* is a row and *X* is a column of *Y*.

— *Y* is a <routine invocation> and *X* is an SQL parameter of *Y*.

— *Y* is a user-defined type instance and *X* is an attribute of *Y*.

— There exists an item *X2* such that *X* is a part of *X2* and *X2* is a part of *Y*.

Another part of the ISO/IEC 9075 series may define additional terms that are used in that part only.

6.3.3.7 Exceptions

Except where otherwise specified, the phrase “an exception condition is raised:”, followed by the name of a condition, is used in General Rules and elsewhere to indicate that the following occurs.

— The execution of a statement is unsuccessful.

— The application of the General Rules of Subclauses are terminated except where explicitly stated otherwise in the General Rules.

— Diagnostic information is to be made available.

— Execution of the statement is to have no effect on SQL-data or schemas.

The effect on assignment targets and SQL descriptor areas of SQL-statements that terminate with exception conditions, unless explicitly defined by the ISO/IEC 9075 series, is implementation-dependent (UA001).

The phrase “a completion condition is raised”, followed by a colon and the name of a condition, is used in General Rules and elsewhere to indicate that application of General Rules is not terminated and dia-

gnostic information is to be made available; unless an exception condition is also raised, the execution of the SQL-statement is successful.

If more than one condition could have occurred as a result of a statement, it is implementation-dependent (UA025) whether diagnostic information pertaining to more than one condition is made available. See Subclause 4.37.2, "Status parameters", in ISO/IEC 9075-2, for rules regarding precedence of status parameter values.

6.3.4 Descriptors

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

Some SQL objects cannot exist except in the context of other SQL objects. For example, columns cannot exist except within the context of tables. Each such object is independently described by its own descriptor, and the descriptor of an enabling object (e.g., table) is said to *include* the descriptor of each enabled object (e.g., column or table constraint). Conversely, the descriptor of an enabled object is said to *be included in* the descriptor of an enabling object. The descriptor of some object *A generally includes* the descriptor of some object *C* if the descriptor of *A* includes the descriptor of *C* or if the descriptor of *A* includes the descriptor of some object *B* and the descriptor of *B* generally includes the descriptor of *C*.

In other cases, certain SQL objects cannot exist unless some other SQL object exists, even though there is no inclusion relationship. For example, SQL does not permit an assertion to exist if some table referenced by the assertion does not exist. Therefore, an assertion descriptor *is dependent on* or *depends on* one or more table descriptors (equivalently, an assertion *is dependent on* or *depends on* one or more tables). In general, a descriptor *D1* can be said to depend on, or to be dependent on, some descriptor *D2*.

The descriptor of some object *A generally depends on* or *is generally dependent on* the descriptor of some object *C* if the descriptor of *A* depends on the descriptor of *C* or if the descriptor of *A* depends on the descriptor of some object *B* and the descriptor of *B* generally depends on the descriptor of *C*.

The execution of an SQL-statement may result in the creation of many descriptors. An SQL object that is created as a result of an SQL-statement may depend on other descriptors that are only created as a result of the execution of that SQL statement.

> NOTE 8 — This is particularly relevant in the case of the <schema definition> SQL-statement. A <schema definition> can, for example, contain many <table definition>s that in turn contain <table constraint>s. A single <table constraint> in one <table definition> can reference a second table being created by a separate <table definition> which itself is able to contain a reference to the first table. The dependencies of each table on the descriptors of the other are valid provided that all necessary descriptors are created during the execution of the complete <schema definition>.

There are two ways of indicating dependency of one SQL object on another. In many cases, the descriptor of the dependent SQL object is said to "include the name of" the SQL object on which it is dependent. In this case "the name of" is to be understood as meaning "sufficient information to identify the descriptor of". Alternatively, the descriptor of the dependent SQL object may be said to include text (e.g., <query expression>, <search condition>) of the SQL object on which it is dependent. However, in such cases, whether the SQL-implementation includes actual text (with defaults and implications made explicit) or its own style of parse tree is irrelevant; the validity of the descriptor is clearly "dependent on" the existence of descriptors of objects that are referenced in it.

In the case of a view, the original <query expression> (not the hierarchical <query expression>) is used to determine the other SQL objects on which the view depends.

> NOTE 9 — The hierarchical <query expression> is used to account for all subtables of a view *V*. If there is a dependency in the hierarchical <query expression> of *V*, this dependency will also be found in the original <query expression> of some subtable of *V*. Destruction of views with dependencies in their original <query expression> will cause the hierarchical <query expression> of the surviving supertables to be rewritten without the dependencies.

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

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

The implementation of some SQL objects described by descriptors requires the existence of objects not specified by the ISO/IEC 9075 series. Where such objects are required, they are effectively created whenever the associated descriptor is created and effectively destroyed whenever the associated descriptor is destroyed.

6.3.5 Application of technical corrigenda

Technical Corrigenda are to be applied to the published text of the parts of the ISO/IEC 9075 series in the following manner.

Where a Technical Corrigendum introduces new Subclauses, the new Subclauses are metaphorically numbered as described here.

— Subclauses inserted between, for example, Subclause 4.3.2 and Subclause 4.3.3 are numbered 4.3.2a, 4.3.2b, etc.

— Those inserted before, for example, 4.3.1 are numbered 4.3.0, 4.3.0a, etc.

Where a Technical Corrigendum introduces new Syntax, Access, General, and Conformance Rules, the new rules are metaphorically numbered as described here.

— Rules inserted between, for example, Rules 7) and 8) are numbered 7.1), 7.2), and so on.

— Those inserted before Rule 1) are numbered 0.1), 0.2), and so on.

Paragraph numbering and the term “lead text” are as described in Subclause 6.3.6.7, “New and modified paragraphs and rules”.

6.3.6 Relationships of parts within the ISO/IEC 9075 series

6.3.6.1 Introduction to relationships among parts

Parts of the ISO/IEC 9075 series other than this document and ISO/IEC 9075-2 depend on this document and ISO/IEC 9075-2 and their Technical Corrigenda and are referenced as *incremental parts*. Each incremental part is to be used as though it were merged with the text of ISO/IEC 9075. This Subclause describes the conventions used to specify the merger. ISO/IEC 9075-2, ISO/IEC 9075-3, ISO/IEC 9075-4, and ISO/IEC 9075-11 are collectively referred to as the *base parts*.

Figure 1, “Relationships between the parts of the ISO/IEC 9075 series” shows the relationships between the various parts of the ISO/IEC 9075 series.

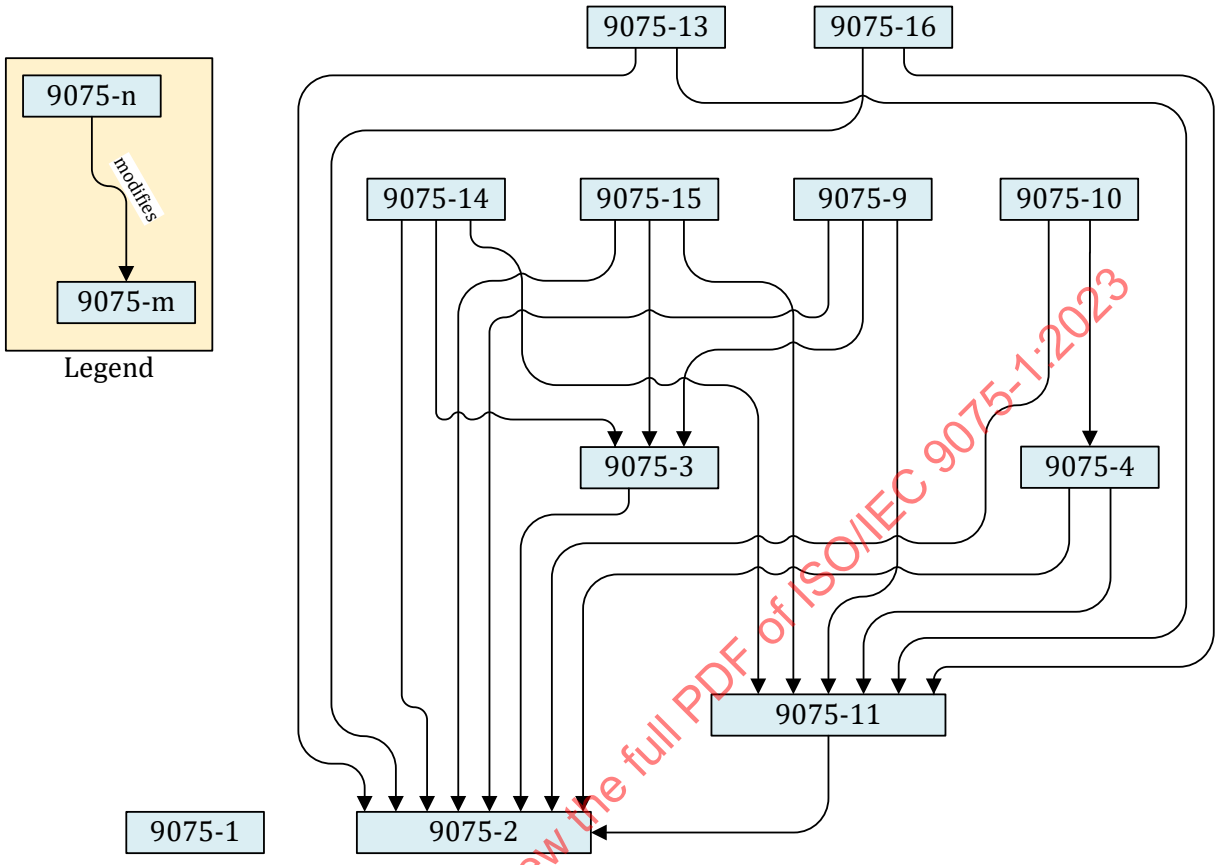

Figure 1 — Relationships between the parts of the ISO/IEC 9075 series

6.3.6.2 Clauses 1, 2, and 3

The text of Clause 1, "Scope" in an incremental part is appended to the text of Clause 1, "Scope", in ISO/IEC 9075-2.

Every entry in Clause 2, "Normative references" in an incremental part that is not already present in Clause 2, "Normative references", in ISO/IEC 9075-2 is added to Clause 2, "Normative references", in ISO/IEC 9075-2.

Every entry in Clause 3, "Terms and definitions" in an incremental part that is not already present in Clause 3, "Terms and definitions", in ISO/IEC 9075-2 is added to Clause 3, "Terms and definitions", in ISO/IEC 9075-2 in a new Subclause of Clause 3, "Terms and definitions" entitled "Definitions taken from ISO/IEC 9075-*n*", where *n* is the number of the incremental part.

6.3.6.3 New and modified Clauses and Subclauses

Where a Clause (other than Clause 1, "Scope", Clause 2, "Normative references", and Clause 3, "Terms and definitions") or Subclause in some incremental part of the ISO/IEC 9075 series has a name identical to a Clause or Subclause in a base part, unless the incremental part references, it supplements the Clause or Subclause, respectively, in the base part, regardless of whether or not the number of the Clause or Subclause corresponds. It typically does so by adding, modifying, or replacing paragraphs, Format items, Tables, Figures, Examples, Equations, Rules, or Notes.

In each incremental part, the relationships between each Clause or Subclause in that incremental part and the corresponding Clause or Subclause in a base part are shown by these considerations.

— In the incremental part: A statement of the form "*This Object modifies Object nn.nn, "xxxxx", in ISO/IEC 9075-n.*" immediately follows the Object title.

— In the part referenced by the statement inserted in the preceding step: A statement of the form "*This Object is modified by Object mm.mmm, "yyyyy", in ISO/IEC 9075-m.*" immediately follows the Object title.

— The Object can be a Clause or a Subclause.

Where a Clause or Subclause in an incremental part has a name that is not identical to the name of some other Clause or Subclause in a base part, it provides language specification particular to that incremental part. A Subclause that is part of a Clause or Subclause identified as new is inherently new and is not marked.

In the absence of explicit instructions regarding its placement, a new Clause is positioned after all existing Clauses, and a new Subclause is positioned after all existing Subclauses of its parent Clause.

6.3.6.4 New and modified Tables, Figures, Examples, and Equations

Tables, figures, examples, and equations are collectively referred to as *captioned items*.

In modified Clauses and Subclauses, where a table in an incremental part of the ISO/IEC 9075 series has a name identical to a table in a base part, unless the incremental part references itself, it supplements the table in the base part, regardless of whether or not the number of the table corresponds.

The rows in tables in the incremental part are generally new rows to be effectively inserted into the corresponding table.

When a captioned item, other than a table, in an incremental part of the ISO/IEC 9075 series has a name identical to a captioned item in a base part, unless the incremental part references itself, it replaces the captioned item in the base part, regardless of whether or not the number of the captioned item corresponds.

When a captioned item in a base part is modified or replaced by a captioned item in some incremental Part, in that incremental part, a statement of the form "*This Object modifies Object nn.nn, "xxxxx", in ISO/IEC 9075-n.*" is typically found in a Rule or paragraph that appears a little above the modifying captioned item.

When a captioned item in base part is modified or replaced by a captioned item in some incremental part, the caption of the captioned item in the base part is preceded by a small red box containing the number of the incremental part that contains the modifying or replacing captioned item.

NOTE 10 — In printed copies of this document, it is possible that the small boxes and their contents appear to be gray or black.

When a captioned item in an incremental part has a name that is not identical to the name of some captioned item in a base part, it provides specifications particular to that incremental part.

6.3.6.5 Functions

In a modified Subclause, the Function from the modifying Subclause completely replaces the Function in the original Subclause.

6.3.6.6 New and modified Format Items

In modified Clauses and Subclauses, a Format item that defines a BNF non-terminal symbol (that is, the BNF non-terminal symbol that appears on the left-hand side of the ::= mark) either modifies a Format

item whose definition appears in a base part, or replaces a Format item whose definition appears in a base part, or defines a new Format item that does not have a definition at all in a base part.

Those Format items in the incremental part that modify a Format item whose definition appears in a base part are identified by the existence of a "Format comment" such as:

```
<modified item> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <new alternative>

<new alternative> ::=
  ...
```

New Format items that have no correspondence to a Format item in a base part are not distinguished in the incremental part.

Format items in new Subclauses are unmarked.

When a Format item in a base part is modified or replaced by a Format item in some incremental Part, the Format item in the base part is indicated by the presence of a small red box, immediately preceding the name of the BNF non-terminal being defined by the Format item, containing the number of the incremental Part that contains the modifying or replacing Format item.

6.3.6.7 New and modified paragraphs and rules

In modified Clauses and Subclauses, each paragraph or Rule is marked to indicate whether it is a modification of a paragraph or Rule in a base part, or is a new paragraph or Rule added by this incremental part.

New paragraphs or Rules in an incremental part are marked to indicate where they are to be inserted.

If no specific insertion point is indicated, as in Insert this paragraph or Insert this GR, then the following text is to be read as though it were appended at the end of the appropriate section (the General Rules, for example) of the corresponding Subclause in the part identified in the statement following the Subclause title.

In such indications, "SR" is used to mean "Syntax Rule", "AR" is used to mean "Access Rule", "GR" is used to mean "General Rule", "CR" is used to mean "Conformance Rule", "Desc." is used to mean "Description", "Func." is used to mean "Function", and "TP" is used to mean "Initial Table Population".

Paragraphs in each Subclause are numbered from 1 (one), and bulleted, dashed, or enumerated lists are counted as part of the same paragraph as the text of the paragraph. Numbered notes are not counted as paragraphs.

When a Rule or paragraph in a base part is modified by a Rule or paragraph in some incremental Part, the Rule or paragraph in the base part is indicated by the presence of a small red box, immediately preceding the text of the Rule or paragraph, containing the number of the incremental Part that contains the modifying Rule or paragraph.

Modifications of paragraphs or Rules are identified by the inclusion of an indicative phrase enclosed in a box.

If the term *lead text of* is included, then the scope of the modification is the the text of the paragraph or rule up to, but not including, the first bulleted, dashed, or enumerated list item that follows. For example, Augment the lead text of SR1)b) means that, in the following rule, the scope of the augmentation is only the italicized text.

1) Lead text for Syntax Rule 1)

    a) Text for Syntax Rule 1)a)

b) *Lead text for Syntax Rule 1)b)*

Case:

i) Text for Syntax Rule 1)b)i)

ii) Text for Syntax Rule 1)b)ii)

c) Text for Syntax Rule 1)c)

Table 3, "Examples and explanations of paragraph and rule mergers", illustrates various ways in which paragraphs and Rules are specified in incremental parts and explains how each is to be interpreted.

Table 3 — Examples and explanations of paragraph and rule mergers

Example	Explanation
Insert into the 1st paragraph, after the 4th list item:	The subsequent text is to be inserted as a new list item immediately after the fourth list item of the unnumbered list that immediately follows the existing first paragraph (effectively becoming a new fifth list item)
Insert after the 2nd paragraph:	The subsequent text is to be inserted as a new paragraph immediately following the existing second paragraph (effectively becoming a new third paragraph)
Insert before SR 4)b):	The subsequent text is to be inserted as a new Syntax Rule immediately preceding the existing SR 4)b) (effectively becoming a new SR 4)b) with subsequent SRs being numbered 4)c), etc.)
Augment the 5th paragraph, the 2nd list item, the 1st list item by adding "new text" immediately after "existing text"	Modify the text of the first unnumbered (bulleted) list item within the first unnumbered (dashed) list item immediately following the existing fifth paragraph by inserting the quoted new text immediately after the quoted existing text
Augment the 3rd paragraph by adding "new text" immediately before "existing text"	Modify the text of the existing third paragraph by inserting the quoted new text immediately before the quoted existing text
Augment SR 8): Add "an SQL variable *SV*" to the list of possible referents of a basis	Augment a list of objects ("possible referents of a basis") in Syntax Rule 8) by adding the specified text to that list
Convert GR 3) to be: Case: a) ... b) Otherwise, the original GR 3)	Convert the exiting GR 3) (which, presumably, has no subrules) into a Case construct, inserting a new subrule, followed by "Otherwise," and the existing text of the existing GR 3)

Example	Explanation

Insert into the Definition after
 	
`FROM ENABLED_ROLES AS ER ) ) ) )`
 	
the code:
 	
`OR`
`( ( R.MODULE_CATALOG, ...`
	Locate the first specified code fragment, then insert the new code fragment immediately following that first fragment

Insert into Table 40, "SQLSTATE class and subclass codes" the rows of Table 3, "SQL-STATE class and subclass codes"	Insert the rows of Table 3 in the current incremental part into the rows of Table 40 in the referenced base part

All paragraphs, Format items, and Rules in new Clauses or Subclauses are also new and are therefore unmarked.

In incremental parts that make modifications to a Subclause in ISO/IEC 9075-2, it is sometimes necessary to replicate a Syntax Rule or a General Rule that redundantly specifies the Subclause signature (see Subclause 6.3.7, "Subclauses used as subroutines") of the modified Subclause in ISO/IEC 9075-2. Doing so permits invocations of the modifying Subclause in the incremental part to correctly specify arguments passed to and results returned from the modified Subclause in ISO/IEC 9075-2. When this occurs, the replication is noted by an instruction such as Replicate GR 1).

6.3.6.8 Modified Subclause Signatures

A Subclause in an incremental part that modifies another part may have a Subclause Signature with additional parameters. These parameters are added to the parameter list of the Signature in the modified part as *optional parameter*s.

6.3.6.9 New and modified Annexes

When an Annex in an incremental part of the ISO/IEC 9075 series has a name identical to an Annex in a base part, unless the incremental part references itself, it supplements the Annex, respectively, in the base part, regardless of whether or not the letter of the Annex corresponds.

Annexes with the titles:

— SQL conformance summary

— implementation-defined elements

— implementation-dependent elements

— SQL optional feature taxonomy

are summaries of certain aspects of normative text in the document. These annexes are never merged, but are effectively regenerated in the merged ISO/IEC 9075-2 document.

In each incremental part, the relationships between each Annex in that incremental part and the corresponding Annex in a base part are shown by the following considerations.

— In the incremental part: A statement of the form "This Annex modifies Annex *N*, "xxxxx", in ISO/IEC 9075-*n*." immediately follows the Annex title.

— In the part referenced by the statement inserted in the preceding step: A statement of the form "This Annex is modified by Annex *M*, "yyyyy", in ISO/IEC 9075-*m*." immediately follows the Annex title.

Where an Annex in an incremental part has a name that is not identical to the name of some other Annex in a base part, it provides information particular to that incremental part.

In a modified Annex, the lead text of the paragraphs of the modifying Annex are discarded.

NOTE 11 — This results in only its Subclauses, tables, and bulleted, dashed, or enumerated list items being retained.

If no explicit merge instructions are provided, then all other text and/or tables are merged into the text and/or tables of the original Annex, taking account of grouping and sorting implied by the initial paragraphs of the original Annex. Redundant duplicate rows in a merged table are eliminated. A row is treated as a duplicate even if the rows have different counters. If explicit merge instructions are given, then these are followed. References to Clauses and Subclauses in the text will be adjusted to the values that the referenced Clauses and Subclauses have acquired in the merged text.

6.3.6.10 Order of merging an incremental part

When an incremental part, after the application of applicable Technical Corrigenda, performs a modification to a Clause or Subclause in a base part, then the modifications are applied in the following sequence:

1) All modifications to the base parts other than ISO/IEC 9075-2 from the incremental part.

2) All modifications to ISO/IEC 9075-11 from other base parts, (including all modifications that were added, augmented, or replaced as a result of Step 1)).

3) All modifications to ISO/IEC 9075-2 from other base parts, (including all modifications that were added, augmented, or replaced as a result of Step 1) and Step 2)).

4) All modifications to ISO/IEC 9075-2 from the incremental part. Modifications in this final step may augment or replace modifications applied as a result of Step 3).

Modifications to one or more base parts by more than one incremental part do not interact. The modifications made by an incremental part only have influence on the language specification of that part and those specifications are not influenced by modifications made by other incremental parts. However, if an SQL-implementation claims conformance to a base part and another incremental part that modifies it, then the modification to the base part from the other incremental part are also applied when applying the merge of the base part.

6.3.7 Subclauses used as subroutines

In this document, some Subclauses are defined without explicit SQL language syntax[4] to invoke their semantics. Such Subclauses, called "subroutine Subclauses", typically factor out rules that are required by one or more other Subclauses and are intended to be invoked by the rules of those other Subclauses. In a few cases, these Subclauses not having explicit SQL language syntax are intended to be invoked by other standards and/or through the use of implementation-defined mechanisms.

In other words, the rules of these Subclauses behave as though they were a sort of definitional "subroutine" that is invoked by other Subclauses in this document, Subclauses in other parts of the ISO/IEC 9075 series, other standards, or implementation-defined mechanisms. These subroutine Subclauses are typically specified in a manner that requires information to be passed to them from their invokers. The information that must be passed is represented as parameters of these subroutine Subclauses, and that information must be passed in the form of arguments provided by the invokers of these subroutine Subclauses. Parameters can be *mandatory parameter*s or *optional parameter*s. Parameters are mandatory by default. Optional parameters are only be created by the merging process as specified in Subclause 6.3.6.8, "Modified Subclause Signatures".

NOTE 12 — There are statements in this document and in other parts of the ISO/IEC 9075 series to the effect of "*something* is valid according to the Syntax Rules of *a Subclause*". Such statements are not equivalent to specifying the invocation of a

4 Some such Subclauses may have non-empty Format sections that define a "sublanguage" that is used in processing SQL-data; that sublanguage is not part of SQL, *per se*, and therefore does not violate this provision.

subroutine Subclause and therefore the referenced Subclause need not be specified in the manner described in the immediately preceding paragraph. Instead, the text identified as *something* is merely evaluated according to the Syntax Rules of the referenced Subclause and determined to be either valid or invalid.

Every invocation of a subroutine Subclause must explicitly provide information for every mandatory parameter of the subroutine Subclause being invoked. Optional parameters that are not specified in a Subclause invocation are assigned the null value. There is an exception to this rule requiring explicit information, which is discussed in Subclause 4.9.3, "Consistency when deleting and updating multiple rows".

In Subclauses that have Subclause Signatures, those signatures are followed by *non-normative* text that documents every parameter and every returned value contained in that signature. That text is intended solely to assist readers of the document in their understanding of the Subclause Signatures and their parameters and returns. If the descriptive text and the normative text in those Subclauses disagree, the Subclauses' normative text is authoritative.

6.3.8 Document typography

In the text of all parts of the ISO/IEC 9075 series, the following typographic conventions are used.

— *Italicized text* is used for several purposes:

- ordinary emphasis;
- representations of SQL truth values (e.g., *True*);
- representations of symbolic variables, both their definitions and their uses;
- textual definitions of important terms and concepts.

— **Bold text** is used for several purposes:

- representations of symbolic variables that contain XML values;
- display of terms taken from other standards that display those terms in bold type.

— Underlined text is used in the representation of SQL truth values (e.g., *True*).

6.3.9 Index typography

In the Indexes to the parts of the ISO/IEC 9075 series, the following conventions are used.

— An index entry in **boldface** indicates the page where the word, phrase, or BNF non-terminal is defined.

— An index entry in *italics* indicates a page where the BNF non-terminal is used in a Format.

— An index entry in neither boldface nor italics indicates a page where the word, phrase, or BNF non-terminal is not defined, but is used other than in a Format (for example, in a heading, Function, Syntax Rule, Access Rule, General Rule, Conformance Rule, Table, or other descriptive text).

6.3.10 Feature ID and Feature Name

Features are either *standard-defined features* or *implementation-defined features*.

Standard-defined features are defined in various parts of the ISO/IEC 9075 series. Implementation-defined features are defined by SQL-implementations (see Subclause 9.5, "Extensions and options").

Features are referenced by a Feature ID and by a Feature Name. A Feature ID value comprises a letter and three digits.

Feature IDs whose letter is "V" are reserved for implementation-defined features.

Standard-defined features are either mandatory features of a part, or they are optional features that are usually defined by Conformance Rules, but in a few circumstances are defined within a paragraph, or within Syntax or General Rules, or are defined as a set of other implied optional features. The Feature ID of a standard-defined feature is stable and can be depended on to remain constant.

For convenience, all of the features defined in a part of the ISO/IEC 9075 series are collected together in a non-normative annex in the part in which they are defined. For example, the features defined in ISO/IEC 9075-2 can be found in Annex D, "SQL optional feature taxonomy", in ISO/IEC 9075-2.

Conformance Rules are generally placed in the Subclause that defines the BNF non-terminal that is controlled by the feature. In those circumstances where the use of a non-terminal is only controlled in a specific circumstance the Conformance Rule is placed where the non-terminal is used. This is also done in those circumstances where the fact that the use of the non-terminal is controlled by the feature can be deduced from an inspection of the Syntax Rules and the Conformance Rules of other Subclauses and the Conformance Rule is in principle redundant. As far as possible, other redundancy in the Conformance Rules is avoided.

6.3.11 Implementation-defined and implementation-dependent

In the ISO/IEC 9075 series, some aspects are marked as "implementation-defined". These aspects are permitted to differ in different implementations but each conforming implementation is required to fully specify that aspect. See Subclause 9.7.3, "Requirements for SQL-implementations", for further information.

Each implementation-defined aspect is assigned a code that is placed in parentheses after each occurrence of that implementation-defined aspect. See for example, 1st paragraph of Subclause 4.4.9.1, "Catalogs":

> A *catalog* is a named collection of SQL-schemas, foreign server descriptors, and foreign data wrapper descriptors in an SQL-environment. The mechanisms for creating and destroying catalogs are implementation-defined (IW005).

The codes assigned to implementation-defined aspects comprise the letter "I" followed by a letter and three digits.

The codes assigned to implementation-defined aspects are stable and can be depended on to remain constant. The codes themselves have no other meaning than to identify the specific implementation-defined aspect.

For convenience, all the implementation-defined aspects in each incremental part of the ISO/IEC 9075 series are collected together in the non-normative Annex B, "Implementation-defined elements", that lists those places in that incremental part where an implementation-defined aspect is referenced.

In the ISO/IEC 9075 series some aspects are marked as "implementation-dependent". These aspects are permitted to differ in different implementations, but that are not necessarily specified for any particular SQL-implementation. Indeed, an SQL-implementation is not required to exhibit consistent behavior with regard to a given aspect. Its behavior may depend on such things as the chosen access path, which may vary over time. An application should not depend on the specific behavior of any implementation-dependent aspects.

Each implementation-dependent aspect is assigned a code that is placed in parentheses after each occurrence of that implementation-dependent aspect. See for example, 2nd paragraph of Subclause 6.3.2, "Specification of the Information and Definition Schemata":

> The only purpose of the view definitions in the Information Schema is to specify the contents of those viewed tables. The actual objects on which these views are based are implementation-dependent (UV002).

The codes assigned to implementation-dependent aspects comprise the letter "U" followed by a letter and three digits.

The codes assigned to implementation-dependent aspects are stable and can be depended on to remain constant. The codes themselves have no other meaning than to identify the specific implementation-dependent aspect.

For convenience, all the implementation-dependent aspects in each incremental part of the ISO/IEC 9075 series are collected together in the non-normative Annex C, "Implementation-dependent elements", that lists those places in that incremental part where an implementation-dependent aspect is referenced.

6.4 Digital artifacts

6.4.1 Introduction to digital artifacts

A number of *digital artifacts* are available for use with the various parts of the ISO/IEC 9075 series. All such digital artifacts are available on the ISO website, links to which are found in each part of the ISO/IEC 9075 series at relevant positions in the text. This Subclause specifies the purpose and suggests uses for each digital artifact.

6.4.2 Language syntax

In the ISO/IEC 9075 series, the grammar (syntax) of database language SQL is defined through the use of an extended BNF notation, as specified in Subclause 6.2, "Notation provided in the ISO/IEC 9075 series".

In each part of the ISO/IEC 9075 series that specifies SQL language grammar, two references to digital artifacts are included. One of those references identifies a plain-text version of the grammar, and the other identifies a version marked up in XML. Implementations of the ISO/IEC 9075 series may use either version when developing language parsers that process the SQL grammar.

The parts of SQL that reference these two digital artifacts, and a reference to the text that incorporates the artifacts are given here.

— ISO/IEC 9075-2 — Subclause 4.1.1, "Notations"
— ISO/IEC 9075-3 — Subclause 4.1.1, "Notations"
— ISO/IEC 9075-4 — Subclause 4.1.1, "Notations"
— ISO/IEC 9075-9 — Subclause 4.1.1, "Notations"
— ISO/IEC 9075-10 — Subclause 4.1.1, "Notations"
— ISO/IEC 9075-13 — Subclause 4.1.1, "Notations"
— ISO/IEC 9075-14 — Subclause 4.1.1, "Notations"
— ISO/IEC 9075-15 — Subclause 4.1.1, "Notations"
— ISO/IEC 9075-16 — Subclause 4.1.1, "Notations"

6.4.3 Condition codes

In the ISO/IEC 9075 series, evaluation of SQL expressions and execution of SQL statements result in the raising of one or more conditions. Conditions are either *completion conditions* or *exception conditions*. Each condition is identified by a unique code, which always includes a *class code* that specifies the principle category to which the condition belongs, and may also include a *subclass code* that specifies additional information about the reason for raising the condition.

In each part of the ISO/IEC 9075 series that specifies language semantics that raises conditions, a summary of the condition codes that are specified is provided. Each such part provides a link to a digital artifact that summarizes all such condition codes. Implementations of ISO/IEC 9075 may use this artifact for

translating the natural language description of each condition (class and subclass codes) to other natural languages for display by relevant software and for documentation purposes.

The parts of SQL that reference this digital artifact, and a reference to the text that incorporates the artifact are given here.

— This document — Subclause 8.1, "SQLSTATE"
— ISO/IEC 9075-2 — Subclause 24.1, "SQLSTATE"
— ISO/IEC 9075-3 — Subclause 10.1, "SQLSTATE"
— ISO/IEC 9075-4 — Subclause 22.1, "SQLSTATE"
— ISO/IEC 9075-9 — Subclause 26.1, "SQLSTATE"
— ISO/IEC 9075-10 — Subclause 18.1, "SQLSTATE"
— ISO/IEC 9075-13 — Subclause 16.1, "SQLSTATE"
— ISO/IEC 9075-14 — Subclause 24.1, "SQLSTATE"
— ISO/IEC 9075-15 — Subclause 20.1, "SQLSTATE"
— ISO/IEC 9075-16 — Subclause 17.1, "SQLSTATE"

6.4.4 Feature codes

In the ISO/IEC 9075 series, a minimum set of features shall be implemented by every SQL-implementation. Other features are optional and SQL-implementations that provide an implementation of optional features may claim conformance to those features. Each such optional feature is identified by a unique code.

In each part of the ISO/IEC 9075 series that specifies optional features, a summary of the optional feature codes that are specified is provided. Each such part provides a link to a digital artifact that summarizes all such feature codes. Implementations of ISO/IEC 9075 may use this artifact for translating the natural language description of each feature to other natural languages for display by relevant software and for documentation purposes.

The parts of SQL that reference this digital artifact, and a reference to the text that incorporates the artifact are given here.

— This document — Annex D, "SQL optional feature taxonomy"
— ISO/IEC 9075-2 — Annex D, "SQL optional feature taxonomy"
— ISO/IEC 9075-3 — Annex D, "SQL optional feature taxonomy"
— ISO/IEC 9075-4 — Annex D, "SQL optional feature taxonomy"
— ISO/IEC 9075-9 — Annex D, "SQL optional feature taxonomy"
— ISO/IEC 9075-10 — Annex D, "SQL optional feature taxonomy"
— ISO/IEC 9075-11 — Annex D, "SQL optional feature taxonomy"
— ISO/IEC 9075-13 — Annex D, "SQL optional feature taxonomy"
— ISO/IEC 9075-14 — Annex D, "SQL optional feature taxonomy"
— ISO/IEC 9075-15 — Annex D, "SQL optional feature taxonomy"
— ISO/IEC 9075-16 — Annex D, "SQL optional feature taxonomy"

6.4.5 Implementation-defined items

In the ISO/IEC 9075 series, a number of details are specified to be *implementation-defined*, as specified in Subclause 7.3, "Implementation-defined elements".

In each part of the ISO/IEC 9075 series that specifies implementation-defined items, a summary of the implementation-defined items is provided. Each such part provides a link to a digital artifact that summarizes all such implementation-defined items. Implementations of ISO/IEC 9075 may use this artifact for translating the natural language description of each such item to other natural languages for display by relevant software, for documentation purposes, and for assisting users in understanding the details of SQL-implementations and in specifying their requirements when deploying SQL-implementations.

The parts of SQL that reference this digital artifact, and a reference to the text that incorporates the artifact are given here.

— This document — Annex B, "Implementation-defined elements"

— ISO/IEC 9075-2 — Annex B, "Implementation-defined elements"

— ISO/IEC 9075-3 — Annex B, "Implementation-defined elements"

— ISO/IEC 9075-4 — Annex B, "Implementation-defined elements"

— ISO/IEC 9075-9 — Annex B, "Implementation-defined elements"

— ISO/IEC 9075-10 — Annex B, "Implementation-defined elements"

— ISO/IEC 9075-11 — Annex B, "Implementation-defined elements"

— ISO/IEC 9075-13 — Annex B, "Implementation-defined elements"

— ISO/IEC 9075-14 — Annex B, "Implementation-defined elements"

— ISO/IEC 9075-15 — Annex B, "Implementation-defined elements"

— ISO/IEC 9075-16 — Annex B, "Implementation-defined elements"

6.4.6 Implementation-dependent items

In the ISO/IEC 9075 series, a number of details are specified to be *implementation-dependent*, as specified in Subclause 7.4, "Implementation-dependent elements".

In each part of the ISO/IEC 9075 series that specifies implementation-dependent items, a summary of the implementation-dependent items is provided. Each such part provides a link to a digital artifact that summarizes all such implementation-dependent items. Implementations of ISO/IEC 9075 may use this artifact for translating the natural language description of each such item to other natural languages for display by relevant software, for documentation purposes, and for assisting users in understanding the details of SQL-implementations.

The parts of SQL that reference this digital artifact, and a reference to the text that incorporates the artifact are given here.

— This document — Annex C, "Implementation-dependent elements"

— ISO/IEC 9075-2 — Annex C, "Implementation-dependent elements"

— ISO/IEC 9075-3 — Annex C, "Implementation-dependent elements"

— ISO/IEC 9075-4 — Annex C, "Implementation-dependent elements"

— ISO/IEC 9075-9 — Annex C, "Implementation-dependent elements"

— ISO/IEC 9075-10 — Annex C, "Implementation-dependent elements"

— ISO/IEC 9075-11 — Annex C, "Implementation-dependent elements"

— ISO/IEC 9075-13 — Annex C, "Implementation-dependent elements"

— ISO/IEC 9075-14 — Annex C, "Implementation-dependent elements"

— ISO/IEC 9075-15 — Annex C, "Implementation-dependent elements"

— ISO/IEC 9075-16 — Annex C, "Implementation-dependent elements"

6.4.7 Header files

Some parts of the ISO/IEC 9075 series provide one or more *header files* that application programs using the facilities of each such part may use to declare necessary symbols and other artifacts.

In each part of the ISO/IEC 9075 series that specifies such header files, the contents of the header files are specified in an Annex. Each such part provides a link to a digital artifact that contains the text of all such header files. Implementations of ISO/IEC 9075 may use these artifacts for documentation purposes, and may supply the text for use by application program writers.

The parts of SQL that reference this digital artifact, and a reference to the text that incorporates the artifact are given here.

— ISO/IEC 9075-3 — Subclause H.1, "C header file sqlcli.h", and Subclause H.2, "COBOL library item SQLCLI".

— ISO/IEC 9075-9 — Subclause H.1, "C header file sqlcli.h", and Subclause H.2, "COBOL library item SQLCLI".

6.4.8 Ada interface

In the ISO/IEC 9075 series, evaluation of SQL expressions and execution of SQL statements result in the raising of one or more conditions, as described in Subclause 6.4.3, "Condition codes".

In each part of the ISO/IEC 9075 series that specifies language semantics that raises conditions and that implements an interface for application programs written in the Ada programming language, the text of (or relevant parts of) an Ada package named Interfaces.SQL is specified. Each such part provides a link to a digital artifact that provides the text of the Ada package or relevant parts of the Ada package. Implementations of ISO/IEC 9075 may use this artifact for documentation purposes and may provide the artifact to application programmers writing programs for such implementations.

The parts of SQL that reference this digital artifact, and a reference to the text that incorporates the artifact are given here.

— ISO/IEC 9075-2 — Subclause 13.3, "<externally-invoked procedure>"

— ISO/IEC 9075-4 — Subclause 12.1, "<externally-invoked procedure>"

— ISO/IEC 9075-9 — Subclause 12.2, "<externally-invoked procedure>"

— ISO/IEC 9075-14 — Subclause 13.1, "<externally-invoked procedure>"

— ISO/IEC 9075-15 — Subclause 12.1, "<externally-invoked procedure>"

— ISO/IEC 9075-16 — Subclause 13.1, "<externally-invoked procedure>"

6.4.9 Schema definition

Some parts of the ISO/IEC 9075 series specify the SQL schema definition and manipulation statements necessary to create the tables, views, and other artifacts of the Definition Schema and Information Schema, described in Subclause 4.4.9.3, "Information Schema", and Subclause 4.4.9.4, "Definition Schema".

In each part of the ISO/IEC 9075 series that specifies such statements, the text of the statements are provided. Each such part provides a link to a digital artifact that contains the text of all such statements. Implementations of ISO/IEC 9075 may use these artifacts for documentation purposes, and may use those statements to create their "databases".

The parts of SQL that reference this digital artifact, and a reference to the text that incorporates the artifact are given here.

— ISO/IEC 9075-4 — Clause 20, "Information Schema", and Clause 21, "Definition Schema".

— ISO/IEC 9075-9 — Clause 24, "Information Schema", and Clause 25, "Definition Schema".

— ISO/IEC 9075-11 — Clause 6, "Information Schema", and Clause 7, "Definition Schema".

— ISO/IEC 9075-13 — Clause 14, "Information Schema", and Clause 15, "Definition Schema".

— ISO/IEC 9075-14 — Clause 21, "Information Schema", and Clause 22, "Definition Schema".

— ISO/IEC 9075-15 — Clause 18, "Information Schema", and Clause 19, "Definition Schema".

— ISO/IEC 9075-16 — Clause 15, "Information Schema", and Clause 16, "Definition Schema".

6.4.10 XML Schemata

Some parts of the ISO/IEC 9075 series specify XML Schema definitions that specifies the XML schemas required by implementations conforming to the specifications of those parts.

In each part of the ISO/IEC 9075 series that specifies such an XML Schema, the text of the XML Schema is provided. Each such part provides a link to a digital artifact that contains the text of such schemas. Implementations of ISO/IEC 9075 may use these artifacts for documentation purposes, and may use those schemas in their implementations, and may provide them to application writers.

The parts of SQL that reference this digital artifact, and a reference to the text that incorporates the artifact are given here.

— ISO/IEC 9075-14 — Clause 23, "SQL/XML XML schema".

7 Annexes to the parts of the ISO/IEC 9075 series

7.1 Annexes are informative

Every annex to every part of the ISO/IEC 9075 series is informative. The contents of each annex provides additional information, some of which restates that which is stated elsewhere in the normative text.

7.2 SQL conformance summary

Every part of the ISO/IEC 9075 series contains an Annex that lists every feature of ISO/IEC 9075, ordered by Feature number, including the number and name of every Subclause that contains a Conformance Rule associated with that feature.

7.3 Implementation-defined elements

Every part of the ISO/IEC 9075 series contains an Annex that lists every element of SQL and its processing that is specified in that part, and is permitted to differ between SQL-implementations, but is required to be specified by the implementor for each particular SQL-implementation.

7.4 Implementation-dependent elements

Every part of the ISO/IEC 9075 series contains an Annex that lists every element of SQL and its processing that is mentioned, but not specified in that part, and is thus permitted to differ between SQL-implementations, but is not required to be specified by the implementor for a particular SQL-implementation.

7.5 Deprecated features

Every part of the ISO/IEC 9075 series contains an Annex that lists every element of SQL and its processing that is specified in that part, but that may be omitted in some future revision of that part.

7.6 Incompatibilities with previous versions

Every part of the ISO/IEC 9075 series contains an Annex that lists every element of SQL and its processing that is specified in a previous version of that part, but that is not specified in the same way in the present version. The most frequent cause of such incompatibilities is the addition of reserved key words to the language, which invalidates their use in SQL language that conformed to an earlier version of ISO/IEC 9075.

7.7 SQL feature taxonomy

ISO/IEC 9075-2 and every incremental part contains an Annex that lists every feature of ISO/IEC 9075 defined in that part, ordered by Feature number, specifying the name of that feature.

7.8 Defect Reports

Every part of the ISO/IEC 9075 series contains an Annex that lists every reported defect in the previous edition of this document that remains in this part of this edition, ordered by Subclause number, specifying the nature of this defect.

8 Status codes

8.1 SQLSTATE

The condition codes and their associated text are available from the ISO website as a "digital artifact". See https://standards.iso.org/iso-iec/9075/-1/ed-6/en/ to download digital artifacts for this document. To download the condition codes and associated text, select the file named ISO_IEC_9075-1(E)_Framework-conditions.xml.

Table 4 — SQLSTATE class and subclass codes

Category	Condition	Class	Subcondition	Subclass
X	*syntax error or access rule violation*	42	(*no subclass*)	000

9 Conformance

9.1 Kinds of conformance claims

Several different types of conformance may be claimed. Every claim of conformance shall include a minimum claim of conformance. In addition, conformance may be claimed for zero or more incremental parts and for zero or more optional features.

9.2 Minimum conformance

Every claim of conformance shall include a *claim of minimum conformance*, which is defined as a claim to meet the conformance requirements specified in ISO/IEC 9075-2 and ISO/IEC 9075-11. A claim of minimum conformance shall include the statements required by the conformance requirements of ISO/IEC 9075-2 and ISO/IEC 9075-11.

Core SQL is set of the features defined in the conformance requirements specified in ISO/IEC 9075-2 and ISO/IEC 9075-11.

9.3 Conformance to parts

A claim of conformance to a part of the ISO/IEC 9075 series implies support of all mandatory features defined in that part.

In addition every claim of conformance to a part of the ISO/IEC 9075 series shall meet the conformance requirements specified in that part.

A claim of conformance to an incremental part of the ISO/IEC 9075 series shall include the claims required by the conformance requirements of that Part of the ISO/IEC 9075 series to which conformance is claimed.

9.4 Conformance to features

In addition to those features that are mandatory for conformance to a part of the ISO/IEC 9075 series (including ISO/IEC 9075-2 and ISO/IEC 9075-11), parts of the ISO/IEC 9075 series may define optional features. These features are identified by Feature ID and controlled by Conformance Rules (see Subclause 6.3.10, "Feature ID and Feature Name").

An optional feature *FEAT* is defined by relaxing selected Conformance Rules, as noted at the beginning of each Conformance Rule by the phrase "without Feature *FEAT*, "name of feature", ...". An application designates a set of SQL features that the application requires; the SQL language of the application shall observe the restrictions of all Conformance Rules except those explicitly relaxed for the required features. Conversely, conforming SQL-implementations shall identify which SQL features the SQL-implementation supports. An SQL-implementation shall process every application whose required features are a subset of the SQL-implementations supported features.

A feature *FEAT1* may imply another feature *FEAT2*. An SQL-implementation that claims to support *FEAT1* shall also support each feature *FEAT2* implied by *FEAT1*. Conversely, an application need only designate that it requires *FEAT1*, and may assume that this includes each feature *FEAT2* implied by *FEAT1*. The list of features that are implied by other features is shown in a table named "Implied feature relationships of ..." in the Clause named "Conformance" in each part of the ISO/IEC 9075 series. Note that some features imply multiple other features, and some features imply features defined in other parts. Every claim of conformance to an optional feature of ISO/IEC 9075 shall meet the conformance requirements of all parts to which conformance is claimed as if the Conformance Rules, which control the feature, did not exist.

The Syntax Rules and General Rules may define one SQL syntax in terms of another. Such transformations are presented to define the semantics of the transformed syntax, and are effectively performed after checking the applicable Conformance Rules, unless otherwise noted in a Syntax Rule that defines a transformation. Transformations may use SQL syntax of one SQL feature to define another SQL feature. These transformations serve to define the behavior of the syntax, and do not have implications for the feature syntax that is permitted or forbidden by the features so defined, except as otherwise noted in a Syntax Rule that defines a transformation. A conforming SQL-implementation need only process the untransformed syntax defined by the Conformance Rules that are applicable for the set of features that the SQL-implementation claims to support, though with the semantics implied by the transformation.

9.5 Extensions and options

An SQL-implementation may provide implementation-defined features that are additional to those specified by ISO/IEC 9075, and may add to the list of reserved words.

NOTE 13 — If additional words are reserved, then it is possible that a conforming SQL-statement will be processed incorrectly.

It is implementation-defined (IA009) whether an SQL Flagger flags implementation-defined features.

NOTE 14 — An SQL Flagger can flag implementation-defined features using any Feature ID not defined by the ISO/IEC 9075 series. However, there is no guarantee that some future edition of the ISO/IEC 9075 series will not use such a Feature ID for a standard-defined feature.

NOTE 15 — The implementation-defined features flagged by an SQL Flagger can include implementation-defined features from more than one SQL-implementation.

NOTE 16 — The allocation of a Feature ID to an implementation-defined feature possibly differ between SQL Flaggers.

It is implementation-defined (IA012) whether an SQL-implementation makes a feature visible through the SQL_FEATURES view of the Information Schema.

It is implementation-defined (IE006) whether an SQL-implementation provides support for additional SQL-invoked routines. It is implementation-defined (IE007) whether an SQL-implementation provides support for additional argument values for SQL-invoked routines.

An SQL-implementation may provide user options to process non-conforming SQL statements or routine invocations. An SQL-implementation may provide user options to process SQL statements or routine invocations so as to produce a result different from that specified in the parts of the ISO/IEC 9075 series.

It shall produce such results only when explicitly required by the user option. Additional extensions and options may be specified in incremental parts of the ISO/IEC 9075 series.

9.6 SQL flagger

An SQL Flagger is a facility, provided by an SQL-implementation, that is able to identify SQL language extensions, or other SQL processing alternatives, that may be provided by a conforming SQL-implementation (see Subclause 9.5, “Extensions and options”).

An SQL Flagger is intended to assist in the production of SQL language that is both portable and interoperable among different conforming SQL-implementations operating under different levels of the ISO/IEC 9075 series.

An SQL Flagger is intended to effect a static check of SQL language. There is no requirement to detect extensions that cannot be determined until the General Rules are evaluated.

An SQL-implementation need only flag SQL language that is not otherwise in error as far as that SQL-implementation is concerned.

NOTE 17 — If a system is processing SQL language that contains errors, then it can be very difficult within a single statement to determine what is an error and what is an extension. As one possibility, an SQL-implementation can choose to check SQL language in two steps; first through its normal syntax analyzer and secondly through the SQL Flagger. The first step produces error messages for non-standard SQL language that the implementation cannot process or recognize. The second step

processes SQL language that contains no errors as far as that SQL-implementation is concerned; it detects and flags at one time all non-standard SQL language that could be processed by that SQL-implementation. Preferably, any such two-step process will be transparent to the end user.

The SQL Flagger assists identification of conforming SQL language that may perform differently in alternative processing environments provided by a conforming SQL-implementation. It also provides a tool in identifying SQL elements that may have to be modified if SQL language is to be moved from a non-conforming to a conforming SQL processing environment.

An SQL Flagger provides one or more of the following "level of flagging" options:

— Core SQL Flagging;

— Part SQL Flagging.

An SQL Flagger that provides one of these options shall be able to identify SQL language constructs that violate the indicated subset of SQL language. The subset of SQL language used by "Core SQL Flagging" is Core SQL. The subset of SQL language used by "Part SQL Flagging" is Core SQL plus those features required for conformance to a specified Part or Parts of the ISO/IEC 9075 series.

An SQL Flagger may also provide "SQL Feature Flagging". "SQL Feature Flagging" indicates which optional features in addition to the indicated subset of SQL language are needed to make the SQL language constructs conforming.

An SQL Flagger provides one or more of the following "extent of checking" options:

— Syntax Only;

— Catalog Lookup.

Under the Syntax Only option, the SQL Flagger analyzes only the SQL language that is presented; it checks for violations of all Syntax Rules that can be determined without access to the Information Schema. It does not necessarily detect violations that depend on the data type of syntactic elements, even if such violations are in principle deducible from the syntax alone.

NOTE 18 — Further details can be found in Annex D, "SQL optional feature taxonomy", in ISO/IEC 9075-2, Annex D, "SQL optional feature taxonomy", in ISO/IEC 9075-3, Annex D, "SQL optional feature taxonomy", in ISO/IEC 9075-4, Annex D, "SQL optional feature taxonomy", in ISO/IEC 9075-9, Annex D, "SQL optional feature taxonomy", in ISO/IEC 9075-10, Annex D, "SQL optional feature taxonomy", in ISO/IEC 9075-11, Annex D, "SQL optional feature taxonomy", in ISO/IEC 9075-13, Annex D, "SQL optional feature taxonomy", in ISO/IEC 9075-14, Annex D, "SQL optional feature taxonomy", in ISO/IEC 9075-15, and Annex D, "SQL optional feature taxonomy", in ISO/IEC 9075-16.

Under the Catalog Lookup option, the SQL Flagger assumes the availability of Definition Schema information and checks for violations of all Syntax Rules. For example, some Syntax Rules place restrictions on data types; this flagger option would identify extensions that relax such restrictions. In order to avoid security breaches, this option shall view the Definition Schema only through the eyes of a specific Information Schema. The flagger does not necessarily execute or simulate the execution of <SQL schema definition statement>s or <SQL schema manipulation statement>s.

An SQL-implementation may claim conformance to Feature F812, "Basic flagging", if and only if it implements an SQL flagger with "level of flagging" specified to be "Core SQL Flagging" and "extent of checking" specified to be "Syntax Only".

An SQL-implementation may claim conformance to Feature F813, "Extended flagging", if and only if it implements an SQL flagger with "level of flagging" specified to be "Core SQL Flagging" and "extent of checking" specified to be "Catalog Lookup".

9.7 Claims of conformance

9.7.1 How conformance is claimed

A claim of conformance to ISO/IEC 9075 shall include all of the following.

— A claim of minimum conformance.

— Zero or more claims of conformance to incremental parts.

— Zero or more claims of conformance to optional features.

NOTE 19 — Each part of the ISO/IEC 9075 series specifies what is to be stated by claims of conformance to that part, in addition to the requirements of this Clause.

9.7.2 Requirements for SQL applications

The term "SQL application" is used here to mean a collection of compilation units that contains one or more of the following.

— SQL statements.

— Invocations of SQL/CLI routines.

— Invocations of externally-invoked procedures.

A conforming SQL application shall be processed without syntax error, provided that all of the following are true.

— Every SQL statement or SQL invocation is syntactically correct in accordance with ISO/IEC 9075.

— The schema contents satisfy the requirements of the SQL application.

— The SQL-data conforms to the schema contents.

— The user has not submitted for immediate execution an SQL-statement that is not syntactically correct.

A conforming SQL application shall not use additional features, or features beyond the level of conformance claimed.

A claim of conformance by an SQL application shall also state the following.

— What implementation-defined elements and actions are relied on for correct performance.

— What schema contents are required to be supplied by the user.

9.7.3 Requirements for SQL-implementations

A conforming SQL-implementation shall process conforming SQL language according to the associated General Rules, Definitions, and Descriptions.

A conforming SQL-implementation shall process conforming routine invocations according to the associated Definitions and General Rules.

A claim of conformance by an SQL-implementation shall also state the following.

— The definition for every element and action, within the scope of the claim, that ISO/IEC 9075 specifies to be implementation-defined.

A conforming SQL-implementation that provides additional facilities or that provides facilities beyond those specified as "Core" shall claim conformance to Feature F812, "Basic flagging".