INTERNATIONAL STANDARD

ISO/IEC 9075-1

First edition
1999-12-01

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 9075-1:1999(E)

PDF disclaimer

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

Adobe is a trademark of Adobe Systems Incorporated.

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

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

Printed in Switzerland

Contents

Page

TABLES

Tables **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.

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

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

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

— Part 1: Framework (SQL/Framework)

— Part 2: Foundation (SQL/Foundation)

— Part 3: Call-Level Interface (SQL/CLI)

— Part 4: Persistent Stored Modules (SQL/PSM)

— Part 5: Host Language Bindings (SQL/Bindings)

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

Introduction

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

1) Clause 1, “Scope”, specifies the scope of this part of ISO/IEC 9075.

2) Clause 2, “Normative references”, identifies additional standards that, through reference in this part of International Standard, constitute provisions of ISO/IEC 9075.

3) Clause 3, “Definitions and use of terms”, defines terms used in this and other parts of ISO/IEC 9075.

4) Clause 4, “Concepts”, describes the concepts used in ISO/IEC 9075.

5) Clause 5, summarises the content of each of the parts of ISO/IEC 9075, in terms of the concepts described in Clause 4, “Concepts”.

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

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

8) Clause 8, specifies requirements that apply to claims of conformance to all or some of the parts of ISO/IEC 9075.

9) Annex A, is an informative Annex. It describes the formal procedures for maintenance and interpretation of ISO/IEC 9075.

10) Annex B, “SQL Packages”, is an informative Annex. It specifies several packages of SQL language features as identified in:

    — Appendix F, "SQL feature and package taxonomy", in ISO/IEC 9075-2

    — Appendix F, "SQL Feature Taxonomy", in ISO/IEC 9075-4

    — Appendix F, "SQL feature and package taxonomy", in ISO/IEC 9075-5

    to which SQL-implementations may claim conformance.

11) Annex C, “Implementation-defined elements”, is an informative Annex. It lists those features for which the body of this part of ISO/IEC 9075 states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or any other behavior is partly or wholly implementation-defined.

12) Annex D, “Implementation-dependent elements”, is an informative Annex. It lists those features for which the body of this part of ISO/IEC 9075 states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or any other behavior is partly or wholly implementation-dependent.

In the text of this part of ISO/IEC 9075, Clauses begin a new odd-numbered page. Any resulting blank space is not significant.

Information technology — Database languages — SQL —

Part 1:
Framework (SQL/Framework)

1 Scope

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

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

NOTE 1 – The coordination of the development of existing and future standards for the management of persistent data in information systems is described by the Reference Model of Data Management (ISO/IEC 10032:1995).

2 Normative references

The following standards contain provisions that, through reference in this text, constitute provisions of this part of ISO/IEC 9075. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 9075 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

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

2) ISO/IEC 9075-2:1999, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation).*

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

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

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

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

7) ISO/IEC CD 14651, *Information technology — International String Ordering — Method for comparing Character Strings.*

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

3 Definitions and use of terms

3.1 Definitions

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

3.1.1 Definitions provided in this standard

In this part of ISO/IEC 9075, the definition of a verb defines every voice, mood, and tense of that verb.

This part of ISO/IEC 9075 defines the following terms, which are also used in other parts of ISO/IEC 9075:

a) **atomic**: Incapable of being subdivided.

b) **compilation unit**: A segment of executable code, possibly consisting of one or more subprograms.

c) **data type**: A set of representable values.

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

e) **identifier**: A means by which something is identified.

f) **identify**: To properly reference something without ambiguity.

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

h) **implementation-dependent**: Possibly differing between SQL-implementations, but not specified by ISO/IEC 9075, and not required to be specified by the implementor for any particular SQL-implementations.

i) **instance (of a value)**: A physical representation of a value. Each instance is at exactly one site. An instance has a data type that is the data type of its value.

j) **null value**: A special value that is used to indicate the absence of any data value.

k) **object (as in "*x* object")**: Any *thing*. An *x* object is a component of, or is otherwise associated with, some *x*, and cannot exist independently of that *x*. 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.

l) **persistent**: Continuing to exist indefinitely, until destroyed deliberately. 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.

m) **property (of an object)**: An attribute, quality, or characteristic of the object.

3.1 Definitions

n) **row**: A sequence of (field name, value) pairs, the data type of each value being specified by the row type.

o) **scope (of a standard)**: The 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.

p) **scope (of a declaration)**: That part of an SQL-client module, SQL-server module, <externally-invoked procedure>, SQL routine, or SQL-statement in which the object declared can be referenced.

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

r) **site**: A place occupied by an instance of a value of some specified data type (or subtype of it).

s) **SQL-connection**: An association between an SQL-client and an SQL-server.

t) **SQL-environment**: The context in which SQL-data exists and SQL-statements are executed.

u) **SQL-implementation**: A processor that processes SQL-statements. A *conforming SQL-implementation* is an SQL-implementation that satisfies the requirements for SQL-implementations as defined in Clause 8, "Conformance".

v) **SQL-session**: The context within which a single user, from a single SQL-agent, executes a sequence of consecutive SQL-statements over a single SQL-connection.

w) **SQL-statement**: A string of characters that conforms, or purports to conform, to the Format and Syntax Rules specified in the parts of ISO/IEC 9075.

x) **table**: A table has an ordered collection of one or more columns and an unordered collection of zero or more rows. Each column has a name and a data type. Each row has, for each column, exactly one value in the data type of that column.

3.2 Use of terms

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

Every object has *properties*, in the usual sense of the word (sometimes called characteristics or attributes), usually including a name that is unique within some class of object. Some objects are dependent on other objects. If *x* is an object, then the objects dependent on it are known as "*x* objects". Thus the term "SQL object" denotes some object that exists only in the context of SQL.

Many *x* objects might be considered to be components of the *x* on which they depend.

If an *x* ceases to exist, then every *x* object dependent on that *x* also ceases to exist.

The representation of an *x* is known as an *x* descriptor or an *x* state, depending on the nature of *x*'s. The descriptor or state of an *x* represents everything that needs to be known about the *x*. See also Subclause 6.2.4, "Descriptors".

3.3 Informative elements

In several places in the body of ISO/IEC 9075, informative notes appear. For example:

NOTE 2 – This is an example of a note.

Those notes do not belong to the normative part ISO/IEC 9075 and conformance to material specified in those notes shall not be claimed.

4 Concepts

4.1 Caveat

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

4.2 SQL-environments and their components

4.2.1 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 catalogs, each of which contains one or more SQL-schemas.

— 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 ISO/IEC 9075, because it has different meanings in the general context.

4.2.2 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 5.6.4, "Direct invocation of SQL"), it is implementation-defined. Alternatively, it may consist of one or more compilation units that, when executed, invoke externally-invoked procedures in an SQL-client module.

4.2.3 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 can 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 vendor and those determined by the installer.

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

4.2.3.1 SQL-clients

An *SQL-client* is a processor, perceived by the SQL-agent as 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.

4.2.3.2 SQL-servers

Each *SQL-server* is a processor, perceived by the SQL-agent as 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.

4.2.4 SQL-client modules

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

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

An SQL-client module contains zero or more externally-invoked procedures.

Exactly one SQL-client module is associated with an SQL-agent at any 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.

4.2.5 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 ISO/IEC 9075.

4.2.6 Catalogs and schemas

4.2.6.1 Catalogs

A *catalog* is a named collection of SQL-schemas in an SQL-environment. The mechanisms for creating and destroying catalogs are implementation-defined.

4.2.6.2 SQL-schemas

An *SQL-schema*, often referred to simply as a *schema*, is a persistent, named collection of descriptors that describe SQL-data. Any 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 mechanisms).

4.2.6.3 The 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 4.2.6.4, “The 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.

4.2.6.4 The 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-data available to an SQL-server. ISO/IEC 9075 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.

4.2.7 SQL-data

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

4.3 Tables

A *table* has an ordered collection of one or more columns and an unordered collection of zero or more rows. Each column has a name and a data type. Each row has, for each column, exactly one value in the data type of that column.

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 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 4.6.6, "Base tables and their components") or a module object (its descriptor is in a module; see Subclause 4.9, "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 base table declared in a module may only be temporary, and is called a *declared 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.

4.4 SQL data types

4.4.1 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 ISO/IEC 9075 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 4.6.4, "User-defined types".

A predefined data type is a data type specified by ISO/IEC 9075, 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.

4.4.2 The 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.

4.4.3 Predefined types

4.4.3.1 Numeric types

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

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. 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.

4.4.3.2 String types

A value of *character 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 maximum. A value of *character large object* (CLOB) type is a string of characters from some character repertoire and is always associated with exactly one character set. A character large object is of variable length, up to some implementation-defined maximum that is probably greater than that of other character strings.

Either a character string or character large object 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 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.

A value of *bit string type* is a string of bits (binary digits). A bit string type is either of fixed length, or of variable length up to some implementation-defined maximum.

A value of *binary string type* (known as a *binary large object*, or BLOB) is a variable length sequence of octets, up to an implementation-defined maximum.

4.4.3.3 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.

4.4.3.4 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.

4.4.3.5 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 a datetime 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.

4.4.4 Constructed atomic types

4.4.4.1 Reference types

A *reference type* is a predefined 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.

4.4.5 Constructed composite types

4.4.5.1 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 may be referenced by their ordinal position in the array.

An array type is specified by an array type constructor.

4.4.5.2 Row types

A row type is a sequence of one or more (field name, data type) pairs, known as fields. A value of a row type consists of one value for each of its fields.

4.4.5.3 Fields

A field is a (field name, data type) pair. A value of the field is a value of its data type.

4.5 Sites and operations on sites

4.5.1 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 be *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 4.6.6.3, "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.

4.5.2 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.

4.5.3 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.

A base table cannot be null, though it may have zero rows.

4.6 SQL-schema objects

4.6.1 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 ISO/IEC 9075 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).

— Character sets.

— Collations.

— Translations.

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 be neither created nor dropped by a user.

4.6.2 Descriptors relating to character sets

4.6.2.1 Character sets

A *character set* is a named set of characters (*character repertoire*) that may be used for forming values of the character data type. Every character set has a *default collation*. Character sets provided by the SQL-implementation, whether defined by other standards or by the 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. 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 *form-of-use*, which specifies the scheme used to collect characters into strings.

This International Standard uses the phrases "character set" and "character repertoire" interchangeably except when referring to data exchanged outside the SQL-implementation, when "character set" is understood to include an encoding and a form-of-use in addition to a character repertoire.

Every character set supported by an SQL-implementation comprises only characters that are represented and expressible using ISO/IEC 10646 UTF-16, the "UCS Transformation Format for Planes of Group 00". This representation is the canonical representation of characters and character strings in this International Standard.

NOTE 3 – ISO/IEC 10646 supports "private use" characters, which are expressible using UTF-16. Future editions of ISO/IEC 10646 are likely to add more characters that are expressible using UTF-16. Such characters are naturally permitted in character sets supported by an SQL-implementation.

4.6.2.2 Collations

A *collation*, also known as a *collating sequence*, is a named operation for ordering character strings in a particular character repertoire. Each collation is defined for exactly one character set.

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

Every collation shall be derived from a collation defined by an International Standard such as ISO/IEC 14561 or by a National Standard, or shall be an implementation-defined collation.

4.6.2.3 Translations

A *translation* 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 translation. 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.

4.6.3 Domains and their components

4.6.3.1 Domains

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

4.6.3.2 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 any value resulting from a cast operation to the domain.

4.6.4 User-defined types

4.6.4.1 Structured types

A *structured type* is a named, user-defined data type. A value of a structured type comprises a number of *attribute values*. Each attribute of a structured type has a data type, specified by an *attribute type* that is included in the descriptor of the structured type.

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. An instance 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*.

4.6.4.2 Attributes

An *attribute* is a named component of a structured type. It has a data type and a default value.

4.6.5 Distinct types

A *distinct type* is a user-defined data type that is based on some data type other than a distinct 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.

4.6.6 Base tables and their components

4.6.6.1 Base tables

A *base table* is a site that holds a table value (see Subclause 4.3, "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*.

4.6.6.2 Columns

A *column* is a named component of a table. It has a data type, a default, and a nullability characteristic.

4.6.6.3 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 any row of the table (but not if it is unknown).

4.6.6.4 Triggers

A *trigger*, though not defined to be a component of a base table, is an object associated with a single base table. A trigger specifies a *trigger event*, a *trigger action time*, and one or more *triggered actions*.

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

A trigger action time specifies whether the triggered action is to be taken BEFORE 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.

4.6.7 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 4.3, “Tables”.

4.6.8 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).

4.6.9 SQL-server modules (defined in ISO/IEC 9075-4, SQL/PSM)

An *SQL-server module* is a module that is a schema object. See Subclause 4.9, "Modules".

4.6.10 Schema routines

A *schema routine* is an SQL-invoked routine that is a schema object. See Subclause 4.10, "Routines".

4.6.11 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 and any that may be added in the future, on 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.

4.6.12 Roles

A *role* is a collection of zero or more role authorizations.

A *role authorization* permits a grantee (see Subclause 4.6.11, "Privileges") to use every privilege granted to the role. It also indicates whether the role authorization is WITH ADMIN OPTION, in which case the grantee is authorized to grant the role.

4.7 Integrity constraints and constraint checking

4.7.1 Constraint checking

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

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 constraint is checked whose constraint mode is immediate.

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

4.7.2 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 spaces 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, and any expression whose value is derived from it, is said to be *possibly non-deterministic* ("possibly", because it may be that all SQL-agents that ever update that column may remove trailing spaces; but this cannot be known to the SQL-implementation).

Because a constraint that contains a possibly non-deterministic expression might be satisfied at one time, yet fail at some later time, no constraint is permitted to contain such an expression.

A routine may claim to be deterministic; if it isn't, then the effect of invoking the routine is implementation-dependent.

4.8 Communication between an SQL-agent and an SQL-implementation

4.8.1 Host languages

An SQL-implementation can communicate successfully with an SQL-agent only if the latter conforms to the standard for some programming language specified by ISO/IEC 9075. Such a language is known generically as a *host language*, and a conforming SQL-implementation is required to support at least one host language.

4.8 Communication between an SQL-agent and an SQL-implementation

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 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 9075-5). In this case, SQL-statements are coded into the application program; an implementation-dependent mechanism is then used to:

  • 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.

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

— Direct invocation of SQL (specified in ISO/IEC 9075-5). 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 that conforms to the standard for the host language.

4.8.2 Parameter passing and data type correspondences

4.8.2.1 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.

4.8.2.2 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.

4.8.2.3 Locators

A host variable, host parameter, SQL parameter or 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.

A locator is an SQL-session object, rather than SQL-data, that can be used to reference an SQL-data instance. A locator is either a large object (LOB) locator, user-defined type locator, or an array locator.

A LOB locator is one of the following:

— Binary large object (BLOB) locator, a value of which identifies a binary large object.

— Character large object (CLOB) locator, a value of which identifies a character large object.

— A national character large object (NCLOB) locator, a value of which identifies a national character large object.

A user-defined type locator identifies a value of the user-defined type specified by the locator specification.

An array locator identifies a value of the array type specified by the locator specification.

When the value at a site of binary large object type, character large object type, user-defined type, or array type is to be assigned to a locator of the corresponding type, an implementation-dependent four-octet non-zero integer value is generated and assigned to the target. A locator value uniquely identifies a value of the corresponding type.

A locator may be either *valid* or *invalid*. A host variable or a host parameter specified as a locator may be further specified as a *holdable locator*. When a locator is initially created, it is marked valid and, if applicable, not holdable. A <hold locator statement> identifying the locator must be specifically executed before the end of the SQL-transaction in which it was created in order to make that locator holdable.

A non-holdable locator remains valid until the end of the SQL-transaction in which it was generated, unless it is explicitly made invalid by the execution of a <free locator statement> or a <rollback statement> that specifies a <savepoint clause> is executed before the end of that SQL-transaction.

A holdable locator may remain valid beyond the end of the SQL-transaction in which it is generated. A holdable locator becomes invalid whenever a <free locator statement> identifying that locator is executed, a <rollback statement> that specifies a <savepoint clause> is executed, or the SQL-transaction in which it is generated or any subsequent SQL-transaction is rolled back. All locators remaining valid at the end of an SQL-session are marked invalid when that SQL-session terminates.

4.8.2.4 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 ISO/IEC 9075. An SQLSTATE value of '00000' (five zeros) indicates that the most recent invocation of an externally-invoked procedure was successful.

4.8.2.5 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 or bit string, and 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.

4.8.3 Descriptor areas (defined in ISO/IEC 9075-5)

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 as 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.

4.8.4 Diagnostic information

A *diagnostics area* is a communication area allocated by the SQL-implementation, that is capable of containing a number of *conditions*. The SQL-agent may specify the size of the area, in terms of conditions, but otherwise the number is one.

Whenever the SQL-implementation returns a status parameter that does not indicate successful completion, it sets values, representing one or more conditions, in the diagnostics area that give some indication of what has happened. These values can be accessed by SQL-diagnostics statements, execution of which do not change the diagnostics area.

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

4.8.5 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* 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 *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 *isolation level* of an SQL-transaction specifies the extent to which the effects of actions by SQL-agents other than 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 UNCOMMITTED and READ COMMITTED, guarantee progressively lower degrees of isolation.

4.9 Modules

There are three 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 4.10, "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.

— An SQL-session module, containing only SQL-statements prepared in that SQL-session.

4.10 Routines

4.10.1 General routine information

Table 1, "Relationships of routine characteristics", shows the terms used for the various possible combinations of SQL/External, SQL-invoked/Externally invoked, and procedures/functions.

Table 1—Relationships of routine characteristics

	SQL routines	**External routines**
SQL-invoked routines	SQL functions and SQL procedures	External functions and procedures
Externally invoked routines	Only SQL procedures (*i.e.*, not functions)	*(not relevant to SQL)*

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

An *externally-invoked routine* is an SQL procedure that is invoked from some compilation unit of a specified standard programming language.

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.

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.

4.10.2 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 ADT, 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.

4.10.3 Built-in functions

A *built-in function*, or *predefined function*, is an SQL-invoked function specified by ISO/IEC 9075. An SQL-implementation may provide additional, implementation-defined, built-in functions.

4.11 SQL-statements

4.11.1 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 ISO/IEC 9075.

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 must, be executed when no SQL-transaction is active.

— According to whether or not they may be embedded in a program written in a standard programming 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 implementations to provide additional, implementation-defined, statements that may fall into any of these categories.

4.11.2 SQL-statements classified by function

The following are the broad classes of SQL-statements:

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

— SQL-data statements, which 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, which set parameters for, and start or terminate transactions.

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

— SQL-connection statements, which 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 set some default values and other parameters of an SQL-session.

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

5 The parts of ISO/IEC 9075

5.1 Overview

This part of ISO/IEC 9075, 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 ISO/IEC 9075 other than parts 1 and 2 is specified as an amendment to ISO/IEC 9075-2. However, the functionality of these other parts is somewhat different in nature.

ISO/IEC 9075-3, Call-level interface, and ISO/IEC 9075-5, Host language bindings, specify mechanisms 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.

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

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

This part of ISO/IEC 9075 contains:

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

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

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

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

ISO/IEC 9075-2 specifies the following features of SQL.

5.3.1 Data types specified in ISO/IEC 9075-2

The following data types are specified in ISO/IEC 9075-2:

— All numeric and string types.

— The Boolean type.

— All datetime and interval types.

— Row types.

— Array types.

— User-defined types.

— Domains.

— Reference types.

5.3.2 Tables

Rules for determining functional dependencies and candidate keys of tables are defined.

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.

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

— One SQL-control statement (RETURN), which can be used to specify a value to be returned by a function.

— 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).

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.

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

ISO/IEC 9075-3 specifies a method of binding between an application program, in one of a number of standard programming languages, and an SQL-implementation. The effect is functionally equivalent to dynamic SQL, specified in ISO/IEC 9075-5 (SQL/Bindings).

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)

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

Those include facilities for:

— 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 local cursors.

— The declaration of local variables.

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

5.5.1 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, including the declaration of handlers to handle exceptions.

— An SQL-control statement (SET PATH), which may be used to control the selection of candidate routines during routine name resolution.

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

5.6 ISO/IEC 9075-5: Host Language Bindings (SQL/Bindings)

ISO/IEC 9075-5 specifies three methods of binding an SQL-agent to an SQL-implementation, and certain facilities for the management of SQL-sessions.

5.6.1 SQL-session facilities

ISO/IEC 9075-5 specifies facilities for setting certain attributes of SQL-sessions that are not specified in ISO/IEC 9075-2. These include default names of catalog, schema, or character set to be used when none is specified.

5.6.2 Dynamic SQL

Dynamic SQL is a method of binding between an application program, in one of a number of standard programming languages, and an SQL-implementation. Facilities are specified to:

— 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.

— Obtain diagnostic information additional to that specified in ISO/IEC 9075-2.

5.6.3 Embedded SQL

Embedded SQL is a method of embedding SQL-statements in a compilation unit that otherwise conforms to the standard 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 that conforms to the particular programming language standard. 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.

5.6.4 Direct invocation of SQL

Direct invocation of SQL is a method of executing SQL-statements directly. In direct invocation of SQL, the following are implementation-defined:

— The method of invoking SQL-statements.

— The method of raising conditions that result from the execution of such statements.

— The method of accessing the diagnostics information that results from the execution of such statements.

— The method of returning the results.

5.6.5 SQL-statements specified in ISO/IEC 9075-5

5.6.5.1 Additional functional classes of SQL-statements

ISO/IEC 9075-5 adds the following classes of SQL-statements:

— 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.

A number of SQL data statements are also added, most of which contain the word "dynamic" in their names. They are not to be confused with SQL-dynamic statements.

For each SQL-statement that it defines, ISO/IEC 9075-5 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 9075-5 specifies whether:

— It may be embedded in a host language.

— It may be dynamically prepared and executed. Any preparable SQL-statement can be executed immediately, with the exception of those that fetch data into a descriptor area.

— It may be executed directly.

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

The notation and conventions defined in this clause are used in the other parts of ISO/IEC 9075, except where more appropriate ones are defined locally.

6.1 Notation

The syntactic notation used in ISO/IEC 9075 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 nonterminal 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 ISO/IEC 9075, the following symbols have the meanings shown:

Symbol	*Meaning*
< >	A character string enclosed in angle brackets is the name of a syntactic element (BNF nonterminal) 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 must be explicitly specified.
|	The alternative operator. The vertical bar indicates that the portion of the formula following the bar is an alternative to the portion preceding the bar. If the vertical bar appears at a position where it is not enclosed in braces or square brackets, it specifies a complete alternative for the element defined by the production rule. If the vertical bar appears in a portion of a formula enclosed in braces or square brackets, it specifies alternatives for the contents of the innermost pair of such braces or brackets.
...	The ellipsis indicates that the element to which it applies in a formula may be repeated any number of times. If the ellipsis appears immediately after a closing brace "}", then it applies to the portion of the formula enclosed between that closing brace and the corresponding opening brace "{". If an ellipsis appears after any other element, then it applies only to that element. 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 normal English text. This is used when the definition of a syntactic element is not expressed in BNF.

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

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

6.1 Notation

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

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

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

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

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

5) Apply steps 1 through 5 to any 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.

6.2 Conventions

6.2.1 Specification of syntactic elements

Syntactic elements are specified in terms of:

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

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

— **Syntax Rules**: A specification 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 must 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 must be supported for conformance to Core SQL.

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

6.2.2 Specification of the Information Schema

The objects of the Information Schema in ISO/IEC 9075 are specified in terms of:

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

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

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

Each Information Schema object is also specified using Conformance Rules that indicate how the view shall be supported for Core 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.

6.2.3 Use of terms

6.2.3.1 Exceptions

Except where otherwise specified (for example, in the General Rules of Subclause 10.4, "<routine invocation>", in ISO/IEC 9075-2), the phrase "an exception condition is raised:", followed by the name of a condition, is used in General Rules and elsewhere to indicate that:

— The execution of a statement is unsuccessful.

— Application of General Rules may be terminated.

— Diagnostic information is to be made available.

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

The effect on any assignment target and SQL descriptor area of an SQL-statement that terminates with an exception condition, unless explicitly defined by ISO/IEC 9075, is implementation-dependent.

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

If more than one condition could have occurred as a result of a statement, it is implementation-dependent whether diagnostic information pertaining to more than one condition is made available. See Subclause 4.26.1, "Status parameters", in ISO/IEC 9075-2, for rules regarding precedence of status parameter values.

6.2.3.2 Syntactic containment

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

6.2 Conventions

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* <C> if *A1* contains *B1* and *A1* contains an instance of <C> that contains *B1*. *A1* is said to contain *B1 without an intervening* <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 <A> contains <B>, then <B> is said to be *contained in* <A> and <A> is said to be a *containing* production symbol for <B>. If <A> simply contains <B>, then <B> is said to be *simply contained in* <A> and <A> is said to be a *simply containing* production symbol for <B>.

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*.

If <A> contains a <table name> that identifies a view that is defined by a <view definition> *V*, then <A> is said to *generally contain* the <query expression> contained in *V*. If <A> contains a <routine invocation> *RI*, then <A> is said to *generally contain* the routine bodies of all <SQL-invoked routine>s in the set of subject routines of *RI*. If <A> contains <B>, then <A> generally contains <B>. If <A> generally contains <B> and <B> generally contains <C>, then <A> generally contains <C>.

NOTE 4 – The "set of subject routines of a <routine invocation>" is defined in Subclause 10.4, "<routine invocation>", in ISO/IEC 9075-2.

6.2.3.3 Terms denoting rule requirements

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

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*.

In the Conformance Rules, the term *shall* defines conditions that are required to be true of SQL language for it to syntactically conform to Core SQL.

6.2.3.4 Rule evaluation order

A conforming 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 implementation.

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

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

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

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

For example, in the production:

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

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

If the result of an expression or search condition 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 is inessential if it is deterministic and does not possibly modify SQL-data; otherwise, it is implementation-defined whether it is essential or inessential.

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

6.2.3.5 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 true 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.

6.2.3.6 Syntactic substitution

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

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

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

Case:

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

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

6.2.3.7 Other terms

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

An SQL-statement *S1* is said to be executed as a *direct result* of 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:

- *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 a part of* an item *Y* if and only if:

- *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 ISO/IEC 9075 may define additional terms that are used in that part only.

6.2.4 Descriptors

A descriptor is a conceptual structured collection of data 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 any descriptor exist in any particular form in any 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 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 be dependent on, some descriptor *D2*.

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 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.

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 be destroyed is included in some other descriptor. Destruction of a descriptor results in the destruction of all descriptors included in it, but has no effect on descriptors on which it is dependent.

The implementation of some SQL objects described by descriptors requires the existence of objects not specified by this International Standard. Where such objects are required, they are effectively created whenever the associated descriptor is created and effectively destroyed whenever the associated descriptor is destroyed.

6.2.5 Relationships of incremental parts to ISO/IEC 9075-2, Foundation

Parts of ISO/IEC 9075 other than this part of ISO/IEC 9075 and ISO/IEC 9075-2 depend on ISO/IEC 9075-2 and its 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.

The merger described also accounts for the Technical Corrigenda that have been published to correct ISO/IEC 9075. This accommodation is typically indicated by the presence of a phrase like "in the Technical Corrigenda" or "in the TC".

6.2.5.1 New and modified Clauses, Subclauses, and Annexes

Where a Clause (other than Clause 1, "Scope", and Clause 2, "Normative references"), Subclause, or Annex in any incremental part of ISO/IEC 9075 has a name identical to a Clause, Subclause, or Annex in ISO/IEC 9075-2 and (unless the incremental part is ISO/IEC 9075-5) ISO/IEC 9075-5, it supplements the Clause, Subclause, or Annex, respectively, in ISO/IEC 9075-2 and/or ISO/IEC 9075-5, regardless of whether or not the number or letter of the Clause, Subclause, or Annex corresponds. It typically does so by adding or replacing paragraphs, Format items, or Rules.

In each incremental part, Table 1, "Clause, Subclause, and Table relationships", identifies the relationships between each Clause, Subclause, and Annex in that incremental part and the corresponding Clause, Subclause, or Annex in ISO/IEC 9075-2 and/or ISO/IEC 9075-5.

Where a Clause, Subclause, or Annex in an incremental part has a name that is not identical to the name of some Clause, Subclause, or Annex in ISO/IEC 9075-2 and/or ISO/IEC 9075-5, it provides language specification particular to that part. A Subclause that is part of a Clause or Subclause identified as new is inherently new and is not marked.

The Clauses, Subclauses, and Annexes in each incremental part appear in the order in which they are intended to appear in the merged document. In the absence of other explicit instructions regarding its placement, any new Clause, Subclause, or Annex is to be positioned as follows: Locate the prior Clause, Subclause, or Annex in ISO/IEC 9075-2 and/or ISO/IEC 9075-5 whose name is identical to the name of a corresponding Clause, Subclause, or Annex that appears in the incremental part of ISO/IEC 9075. The new Clause, Subclause, or Annex shall immediately follow that Clause, Subclause, or Annex. If there are multiple new Clauses, Subclauses, or Annexes with no intervening Clause, Subclause, or Annex that modifies an existing Clause, Subclause, or Annex, then those new Clauses, Subclauses, or Annexes appear in order, following the prior Clause, Subclause, or Annex whose name was matched.

When an incremental part performs a modification to the Clause, Subclause, or Annex in ISO/IEC 9075-5, then the modifications are applied in the following sequence:

1) All modifications to ISO/IEC 9075-5 from the incremental part.

2) All modifications to ISO/IEC 9075-2 from ISO/IEC 9075-5, including all modifications that were added, augmented, or replaced as a result of step 1.

3) All modifications to ISO/IEC 9075-2 from the incremental part. Note that modifications in this third step may augment or replace modifications applied as a result of the first two steps.

Modifications to ISO/IEC 9075-2 and/or ISO/IEC 9075-5 by more than one incremental part to 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 any other incremental part.

6.2.5.2 New and modified Format items

In a modified Subclause, a Format item that defines a BNF nonterminal symbol (that is, the BNF nonterminal symbol appears on the left-hand side of the ::= mark) either modifies a Format item whose definition appears in ISO/IEC 9075-2 and/or ISO/IEC 9075-5, or replaces a Format item whose definition appears in ISO/IEC 9075-2 and/or ISO/IEC 9075-5, or defines a new Format item that does not have a definition at all in ISO/IEC 9075-2 and/or ISO/IEC 9075-5. Those Format items in the incremental part that modify a Format item whose definition appears in ISO/IEC 9075-2 and/or ISO/IEC 9075-5 are identified by the existence of a "Format comment" such as:

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

By contrast, Format items that completely replace Format items in ISO/IEC 9075-2 and/or ISO/IEC 9075-5 have BNF nonterminal symbols identical to BNF nonterminal symbols of Format items in ISO/IEC 9075-2 and/or ISO/IEC 9075-5, but do not state that they include any alternatives from ISO/IEC 9075-2 and/or ISO/IEC 9075-5.

New Format items that have no correspondence to any Format item in ISO/IEC 9075-2 and/or ISO/IEC 9075-5 are not distinguished in the incremental part.

Format items in new Subclauses are unmarked.

6.2.5.3 New and modified paragraphs and rules

In modified Subclauses, each paragraph or Rule is marked to indicate whether it is a modification of a paragraph or Rule in ISO/IEC 9075-2, or is a new paragraph or Rule added by this incremental part.

Modifications of paragraphs or Rules in ISO/IEC 9075-2 are identified by the inclusion of an indicative phrase enclosed in a box.

Replace the 5th paragraph means that the following text is to replace the fifth paragraph of the corresponding Subclause in ISO/IEC 9075-2.

Replace SR6)b)ii) means that the following text is to replace Syntax Rule 6)b)ii) of the corresponding Subclause in ISO/IEC 9075-2.

Augments SR3) means that the following text is to extend or enhance Syntax Rule 3). In most instances, the augmentation is the addition of a new alternative meant to support new syntax.

New paragraphs or Rules in an incremental part is marked to indicate where it is to be inserted.

Insert before 2nd paragraph means that the following text is to be read as though it were inserted immediately before the second paragraph of the corresponding Subclause in ISO/IEC 9075-2.

Insert before GR4) means that the following text is to be read as though it were inserted immediately before General Rule 4) of the corresponding Subclause in ISO/IEC 9075-2.

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 ISO/IEC 9075-2.

Modifications of paragraphs or Rules in ISO/IEC 9075-5 are identified in the same way as for modifications of ISO/IEC 9075-2, except that "in Part 5" is appended to the indicative phrase.

In such indications, “SR” is used to mean “Syntax Rule”, “AR” is used to mean “Access Rule”, “GR” is used to mean “General Rule”, and “CR” is used to mean “Conformance Rule”. “Desc.” is used to mean “Description” and “Func.” is used to mean “Function”.

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

6.2.5.4 New and modified tables

If the name of a table in an incremental part is identical to that of a table in ISO/IEC 9075-2 and/or ISO/IEC 9075-5, then the table supplements the table in ISO/IEC 9075-2 and/or ISO/IEC 9075-5, typically by adding or replacing one or more table entries; otherwise, it is a new table.

In each incremental part, there is a table, Table 1, "Clause, Subclause, and Table relationships", that identifies the relationships between tables in that incremental part and the corresponding tables in ISO/IEC 9075-2 and/or ISO/IEC 9075-5,

The rows in modified tables are generally new rows to be effectively inserted into the corresponding table in ISO/IEC 9075-2 and/or ISO/IEC 9075-5, though in rare cases a row already in a table in ISO/IEC 9075-2 and/or ISO/IEC 9075-5 is effectively replaced by a row in the table in the incremental part. Such replacement is required wherever the value in the first column of the corresponding table is the same.

6.2.6 Index typography

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

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

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

— An index entry in neither boldface nor italics indicates a page where the word, phrase, or BNF nonterminal 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 Object identifier for Database Language SQL

Database language SQL has an object identifier, defined using the facilities of ISO/IEC 8824-1, that identifies the characteristics of an SQL-implementation. Each part of ISO/IEC 9075 other than ISO/IEC 9075-2 specifies the content of the Object Identifier for that part.

Function

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

NOTE 5 – The equivalent information is available to the SQL user in the Information Schema.

Format

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

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

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

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

<arc3> ::= 9075

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

<SQL edition> ::= <1987> | <1989> | <1992> | <1999>

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

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

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

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

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

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

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

<SQL conformance> ::= <level> <parts> <packages>

<level> ::= <low> | <intermediate> | <high>

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

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

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

<1999> ::= 3| edition1999 <left paren> 3 <right paren>

<parts> ::= <Part 3> <Part 4> <Part 5> <Part 6> <Part 7> <Part 8> <Part 9> <Part 10>

<Part n> ::= <Part n no> | <Part n yes>

<Part n no> ::= 0 | Part-nNo <left paren> 0 <right paren>

<Part n yes> ::= !! as specified in ISO/IEC 9075-n

<packages> ::=
      <Package PKGi>...

<Package PKGi> ::=
        <Package PKGiYes>
      | <Package PKGiNo>
```

NOTE 6 – For $n \geq 3$, <Part *n* yes> is specified in part *n* of ISO/IEC 9075.

Syntax Rules

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

2) If <SQL edition> is <1999>, then <level> shall not be specified, <parts> shall be specified, and conformance is claimed to Core SQL; otherwise, <level> shall be specified and <parts> shall not be specified.

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

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

      Case:

      i) <low>, then Entry SQL level.

      ii) <intermediate>, then Intermediate SQL level.

      iii) <high>, then Full SQL level.

   b) Otherwise,

      Case:

      i) <low>, then level 1 (one).

      ii) <intermediate>, then level 2.

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

5) <parts> shall not be specified unless <SQL edition> is <1999>; <level> shall not be specified if <SQL edition> is <1999>.

6) <packages> shall not be specified unless <SQL edition> is <1999>.

7) <Package PKG*i*Yes>, where PKG*i* identifies an optional package described in this part of ISO/IEC 9075 or in some profile of ISO/IEC 9075, implies that conformance to the optional package identified by PKG*i* is claimed; <Package PKG*i*No> implies that conformance to the optional package identified by PKG*i* is not claimed.

8) Specification of <Part *n* No> implies that conformance to ISO/IEC 9075-*n* is not claimed.

7 Annexes to the parts of ISO/IEC 9075

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

7.1 Implementation-defined elements

Every part of ISO/IEC 9075 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.

7.2 Implementation-dependent elements

Every part of ISO/IEC 9075 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 any particular SQL-implementation.

7.3 Deprecated features

ISO/IEC 9075-2 and every incremental part contains an Annex that lists every element of SQL and its processing that is specified in that part, but that may not be specified in some future revision of that part.

7.4 Incompatibilities with previous versions

ISO/IEC 9075-2 and every incremental part 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.

8 Conformance

8.1 Requirements for SQL-implementations

An SQL-implementation shall support Core SQL and at least one of the following:

— The SQL-client module binding, as specified in ISO/IEC 9075-2, for at least one host language.

— Embedded SQL, as specified in ISO/IEC 9075-5, for at least one host language.

8.1.1 Parts and packages

An SQL-implementation may support the requirements of any incremental part of ISO/IEC 9075.

The SQL-implementation shall provide an Object Identifier (defined in Subclause 6.3, "Object identifier for Database Language SQL") that states the parts of ISO/IEC 9075 for which conformance is claimed.

For each additional part of ISO/IEC 9075 for which conformance is claimed, the SQL-implementation shall comply with all conformance requirements specified in that part.

An SQL-implementation may additionally support the requirements of one or more packages of features of ISO/IEC 9075, as specified in Annex B, "SQL Packages". For each package for which conformance is claimed, the SQL-implementation shall comply with all conformance requirements specified for that package.

8.1.2 Functionality

For each part of ISO/IEC 9075 for which conformance is claimed, an SQL-implementation

— Shall process every SQL-statement according to the applicable rules.

— Shall provide and maintain an Information Schema for each catalog.

8.1.3 Additional features

An SQL-implementation may provide features additional to those specified by Core SQL, additional parts of ISO/IEC 9075 to which conformance is claimed, and any packages to which conformance is claimed, and may add to the list of reserved words.

NOTE 7 – If additional words are reserved, it is possible that a conforming SQL-statement may not be processed correctly.

An SQL-implementation may provide user options to process non-conforming SQL statements.

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

It shall produce such results only when explicitly required by the user option.

An SQL-implementation that provides additional facilities or that provides facilities beyond those specified as "Core" shall provide an SQL-Flagger.

8.1.4 SQL flagger

An SQL Flagger is an implementation-provided facility that is able to identify SQL language extensions, or other SQL processing alternatives, that may be provided by a conforming SQL-implementation (see Subclause 8.1.3, "Additional features").

An SQL Flagger is intended to assist SQL programmers in producing SQL language that is both portable and interoperable among different conforming SQL-implementations operating under different levels of this International Standard.

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

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

NOTE 8 – If a system is processing SQL language that contains errors, then it may be very difficult within a single statement to determine what is an error and what is an extension. As one possibility, an implementation may choose to check SQL language in two steps; first through its normal syntax analyzer and secondly through the 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 implementation is concerned; it detects and flags at one time all non-standard SQL language that could be processed by that implementation. Any such two-step process should be transparent to the end user.

The SQL Flagger allows an application programmer to identify conforming SQL language that may perform differently in alternative processing environments provided by a conforming SQL-implementation. It also provides a 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

— Package 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. "Core SQL Flagging" indicates those features to be found in Core SQL as defined in the Conformance Rules of ISO/IEC 9075-2 and 9075-5. "Part SQL Flagging" indicates those features defined in a specified Part or Parts of ISO/IEC 9075 other than 9075-2 and 9075-5. "Package Flagging" indicates those features defined as being part of a specified Package or Packages of ISO/IEC 9075.

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

— Syntax Only

— Catalog Lookup

Under the Syntax Only option, the SQL Flagger analyzes only the SQL language that is presented; it checks for violations of any Syntax Rules that can be determined without access to the Information Schema. 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.

Under the Catalog Lookup option, the SQL Flagger assumes the availability of Definition Schema information and checks for violations of all Syntax Rules and Access Rules, except Access Rules that deal with privileges. For example, some Syntax Rules place restrictions on data types; this flagger option would identify extensions that relax such restrictions. In order to avoid security breaches, this option 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 any <schema definition statement> or <schema manipulation statement>.

8.1.5 Claims of conformance

A claim of conformance to one or more parts of ISO/IEC 9075 shall include:

1) A list of those parts to which conformance is claimed.

2) The definition for every element and action that ISO/IEC 9075 specifies to be implementation-defined.

NOTE 9 – Each part of ISO/IEC 9075 specifies what shall be stated by claims of conformance to that part, in addition to the requirements of this clause.

8.2 Requirements for SQL applications

8.2.1 Introduction

The term “SQL application” is used here to mean a collection of compilation units, each in some standard programming language, that contains one or more of:

— SQL statements.

— Invocations of SQL/CLI routines.

— Invocations of externally invoked procedures in SQL-client modules.

8.2.2 Requirements

A conforming SQL application shall be processed without syntax error, provided:

— Every SQL statement or SQL/CLI 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 a syntactically erroneous SQL statement.

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

8.2.3 Claims of conformance

A claim of conformance by an SQL application shall state:

— What incremental parts of ISO/IEC 9075 are required to be supported.

— What implementation-defined features are relied on for correct performance.

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