SC32 N00001

ISO Final Committee Draft (FCD)

Database Language SQL — Part 1: SQL/Framework

«Part 1»

October 1997

Contents

Page

Foreword vii

Introduction ix

1 Scope 1

2 Normative references 3

3 Definitions and use of terms 5
3.1 Definitions 5
3.1.1 Definitions provided in this standard 5
3.2 Use of terms 6
3.3 Informative elements 6

4 Concepts 7
4.1 Caveat 7
4.2 SQL-environments and their components 7
4.2.1 SQL-environments 7
4.2.2 SQL-agents 7
4.2.3 SQL-implementations 7
4.2.3.1 SQL-clients 8
4.2.3.2 SQL-servers 8
4.2.4 SQL-client modules 8
4.2.5 Authorization identifiers 8
4.2.6 Catalogs and schemas 8
4.2.6.1 Catalogs 9
4.2.6.2 SQL-schemas 9
4.2.6.3 The Information Schema 9
4.2.6.4 The Definition Schema 9
4.2.7 SQL-data 9
4.3 Tables 10
4.4 SQL data types 10
4.4.1 General 10
4.4.2 The null value 11
4.4.3 Predefined atomic types 11
4.4.3.1 Numeric types 11
4.4.3.2 String types 11
4.4.3.3 Boolean type 11
4.4.3.4 Datetime types 12
4.4.3.5 Interval types 12
4.4.3.6 Reference types 12

4.4.4 Predefined composite types . . . 12
4.4.4.1 Collection types . . . 12
4.4.4.2 Row types . . . 12
4.4.4.3 Fields . . . 13
4.4.5 Locator types . . . 13
4.5 Sites and operations on them . . . 13
4.5.1 Sites . . . 13
4.5.2 Assignment and mutation . . . 14
4.5.3 Nullability . . . 14
4.6 SQL-schema objects . . . 14
4.6.1 General . . . 14
4.6.2 Descriptors relating to character sets . . . 15
4.6.2.1 Character sets . . . 15
4.6.2.2 Collations . . . 15
4.6.2.3 Translations . . . 15
4.6.3 Domains and their components . . . 15
4.6.3.1 Domains . . . 15
4.6.3.2 Domain constraints . . . 15
4.6.4 Abstract data types and their components . . . 16
4.6.4.1 Abstract data types . . . 16
4.6.4.2 Attributes . . . 16
4.6.5 Distinct types . . . 16
4.6.6 Named row types . . . 16
4.6.7 Base tables and their components . . . 16
4.6.7.1 Base tables . . . 16
4.6.7.2 Columns . . . 17
4.6.7.3 Table constraints . . . 17
4.6.7.4 Triggers . . . 17
4.6.8 View definitions . . . 17
4.6.9 Assertions . . . 18
4.6.10 SQL-server modules (defined in ISO/IEC 9075-4, SQL/PSM) . . . 18
4.6.11 Schema routines . . . 18
4.6.12 Privileges . . . 18
4.6.13 Roles . . . 18
4.7 Integrity constraints and constraint checking . . . 18
4.7.1 Constraint checking . . . 18
4.7.2 Determinism and constraints . . . 19
4.8 Communicating between the SQL-agent and the SQL-implementation . . . 19
4.8.1 Host languages . . . 19
4.8.2 Parameter passing and data type correspondences . . . 20
4.8.2.1 General . . . 20
4.8.2.2 Data type correspondences . . . 20
4.8.2.3 Status parameters . . . 20
4.8.2.4 Indicator parameters . . . 20
4.8.3 Descriptor areas (defined in ISO/IEC 9075-5) . . . 20

4.8.4 Diagnostic information 21
4.8.5 SQL-Transactions 21
4.9 Modules 21
4.10 Routines 22
4.10.1 General 22
4.10.2 Identity functions 22
4.10.3 Built-in functions 22
4.11 SQL-statements 23
4.11.1 Classes of SQL-statements 23
4.11.2 SQL-statements classified by function 23

5 The parts of ISO/IEC 9075 25
5.1 Overview 25
5.2 ISO/IEC 9075-1: Framework (SQL/Framework) 25
5.3 ISO/IEC 9075-2: Foundation (SQL/Foundation) 25
5.3.1 Data types specified in ISO/IEC 9075-2 25
5.3.2 Tables 26
5.3.3 SQL-statements specified in ISO/IEC 9075-2 26
5.4 ISO/IEC 9075-3: Call Level Interface (SQL/CLI) 26
5.5 ISO/IEC 9075-4: Persistent Stored Modules (SQL/PSM) 27
5.5.1 SQL-statements specified in ISO/IEC 9075-4 27
5.6 ISO/IEC 9075-5: Host Language Bindings (SQL/Bindings) 28
5.6.1 SQL-session facilities 28
5.6.2 Dynamic SQL 28
5.6.3 Embedded SQL 28
5.6.4 Direct invocation of SQL 28
5.6.5 SQL-statements specified in ISO/IEC 9075-5 29
5.6.5.1 Additional functional classes of SQL-statements 29
5.7 ISO/IEC 9075-6: XA Specialization (SQL/Transaction) 29
5.8 ISO/IEC 9075-7: Temporal (SQL/Temporal) 29

6 Notation and conventions used in other parts of ISO/IEC 9075 31
6.1 Notation 31
6.2 Conventions 32
6.2.1 Specification of syntactic elements 32
6.2.2 Specification of the Information Schema 33
6.2.3 Use of terms 33
6.2.3.1 Exceptions 33
6.2.3.2 Syntactic containment 33
6.2.3.3 Terms denoting rule requirements 34
6.2.3.4 Rule evaluation order 35
6.2.3.5 Conditional rules 35
6.2.3.6 Syntactic substitution 36
6.2.3.7 Other terms 36
6.2.4 Descriptors 37
6.2.5 Relationships of incremental parts to ISO/IEC 9075-2, Foundation 38

6.2.5.1 New and modified Clauses, Subclauses, and Annexes 38
6.2.5.2 New and modified Format items 38
6.2.5.3 New and modified paragraphs and rules 39
6.2.5.4 New and modified tables 39
6.2.6 Index typography 40
6.3 Object identifier for Database Language SQL 40

7 Annexes to the parts of ISO/IEC 9075 43
7.1 Implementation-defined elements 43
7.2 Implementation-dependent elements 43
7.3 Deprecated features 43
7.4 Incompatibilities with previous versions 43

8 Conformance 45
8.1 Requirements for SQL-implementations 45
8.1.1 Parts and packages 45
8.1.2 Functionality 45
8.1.3 Additional features 45
8.1.4 Claims of conformance 46
8.2 Requirements for SQL applications 46
8.2.1 Introduction 46
8.2.2 Requirements 46
8.2.3 Claims of conformance 47

Annex A Maintenance and interpretation of SQL 49

Annex B SQL Feature Taxonomy 51

Annex C SQL Packages 65
C.1 Enhanced datetime facilities 65
C.2 Enhanced integrity management 65
C.3 OLAP facilities 66
C.4 PSM 66
C.5 CLI 66
C.6 Basic object support 66
C.7 Enhanced object support 67

Index Index1

TABLES

Table		Page
1	Relationships of routine characteristics	22
2	SQL/Foundation feature taxonomy	52
3	SQL/Bindings feature taxonomy	64

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 was prepared by Joint Technical Committee ISO/IEC JTC 1, Information Technology.

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

— Part 1: Framework (SQL/Framework) describes the fundamental concepts on which specifications in other parts of ISO/IEC 9075 are based. It also defines terms, notations and conventions used in ISO/IEC 9075. It specifies general requirements for conformance.

— Part 2: Foundation (SQL/Foundation) specifies the fundamentals of SQL.

— Part 3: Call-Level Interface (SQL/CLI) specifies an interface to SQL that may be used by an application program.

— Part 4: Persistent Stored Modules (SQL/PSM) specifies control structures that may be used to define SQL-routines, and the modules that may contain them.

— Part 5: Host Language Bindings (SQL/Bindings) specifies how SQL-statements may be embedded in programs in a host programming language, and how SQL-statements may be prepared for execution and executed.

— Part 6: XA Specialization (SQL/Transaction) specifies how SQL may be used with a standard conforming transaction manager.

— Part 7: Temporal (SQL/Temporal) specifies facilities for defining and manipulating temporal data.

Parts other than part 1 specify requirements, and all are dependent on part 1. Parts other than parts 1 and 2 are all dependent on part 2.

Introduction

The organization of ISO/IEC 9075-1 is as follows:

1) Clause 1, "Scope", specifies the scope of ISO/IEC 9075-1.

2) Clause 2, "Normative references", identifies additional standards that, through reference in this International Standard, constitute provisions of ISO/IEC 9075-1.

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. describes the formal procedures for maintenance and interpretation of ISO/IEC 9075.

10) Annex B, "SQL Feature Taxonomy", is an informative Annex. It identifies features of the SQL language specified in this part of ISO/IEC 9075 by a numeric identifier and a short descriptive name. This taxonomy is used to specify conformance to Core SQL and may be used to develop other profiles involving the SQL language.

11) Annex C, "SQL Packages", is an informative Annex. It specifies several packages of SQL language features (as identified in Annex B, "SQL Feature Taxonomy") to which SQL-implementations may claim conformance.

In the text of ISO/IEC 9075-1, Clauses begin a new odd-numbered page. Any resulting blank space is not significant.

Information technology — Database languages — SQL — Part 1: SQL/Framework

1 Scope

ISO/IEC 9075-1 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.

ISO/IEC 9075-1 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 ISO/IEC 9075-1. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on ISO/IEC 9075-1 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 9075-2:199*x*, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation).*

ISO/IEC 9075-3:199*x*, *Information technology — Database languages — SQL — Part 3: Call-level interface (SQL/CLI).*

ISO/IEC 9075-4:199*x*, *Information technology — Database languages — SQL — Part 4: Persistent stored modules (SQL/PSM).*

ISO/IEC 9075-5:199*x*, *Information technology — Database languages — SQL — Part 5: Host language bindings (SQL/Bindings).*

ISO/IEC 9075-6:199*x*, *Information technology — Database languages — SQL — Part 6: XA Specialization (SQL/Transaction).*

ISO/IEC 9075-7:199*x*, *Information technology — Database languages — SQL — Part 7: Temporal (SQL/Temporal).*

3 Definitions and use of terms

3.1 Definitions

For the purposes of ISO/IEC 9075-1, the following definitions apply.

3.1.1 Definitions provided in this standard

In ISO/IEC 9075-1, the definition of a verb defines every voice, mood, and tense of that verb.

ISO/IEC 9075-1 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) **implementation-defined**: Possibly differing between SQL-implementations, but specified by the implementor for each particular SQL-implementation.

f) **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.

g) **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.

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

i) **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.

j) **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.

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

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

m) **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.

n) **scope (of a declaration)**: That part of an SQL-statement in which the object declared can be referenced.

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

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

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

r) **SQL-implementation**: A processor that processes SQL-statements. A *conforming SQL-implementation* is an SQL-implementation that conforms to

s) **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.

t) **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 which 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", below.

3.3 Informative elements

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

NOTE 2 – This is an example of a note.

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

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 be 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 and the SQL-agent.

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 (*q.v.*) 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 Authorization identifiers

An *authorization identifier* represents a user. The means of creating and destroying authorization 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 one or more catalogs available to an SQL-session are known as a *cluster*, and contain the descriptors that describe all the SQL-data accessible through that SQL-server.

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 Schemas in the cluster of catalogs at that SQL-server.

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*.

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 or viewed table can have the same name, though this constraint does not apply to tables in general.

A base table is either a schema object (its descriptor is in a schema, see Subclause 4.6.7, "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 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

Every data value belongs to some data type.

Every data type is either *predefined* or *user-defined*. Every data type has a name. The name of a predefined data type is a reserved word specified by that part of ISO/IEC 9075 which specifies the data type. A user-defined data type is a schema object, see Subclause 4.6.4, "Abstract data types and their components", Subclause 4.6.5, "Distinct types", and Subclause 4.6.6, "Named row 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 either atomic or composite. A predefined 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 predefined 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.

— In some contexts the null value is treated as neither equal nor unequal to any value, including itself. In others it is treated as equal to itself and unequal to all other values.

4.4.3 Predefined atomic 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 scale 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). Except when overflow occurs, arithmetic operations on operands of different precisions and scales do not normally cause problems. If a value cannot be represented exactly, then whether it is rounded or truncated is implementation-defined.

4.4.3.2 String types

A value of *character type* is a string (sequence) of characters from some character repertoire. 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 of variable length, up to an 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 *national character repertoire*, by specifying NATIONAL in 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 zero 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 zero 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.

4.4.3.6 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 base tables that have the *with REF value* property. Since this property is permitted only on base tables of named row type (see Subclause 4.6, "SQL-schema objects", below), it follows that every referenced type is a named row type.

4.4.4 Predefined composite types

4.4.4.1 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.

4.4.4.2 Row types

A row type is a sequence of (field name, data type) pairs, known as fields.

4.4.4.3 Fields

A field is a (field name, data type) pair.

4.4.5 Locator types

An embedded variable, parameter, or function result may be specified to be of a *locator type*. The purpose of a locator is to allow very large data instances to be operated on without transferring the whole of them 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, an ADT locator, or an array locator. A LOB locator one of the following:

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

— 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.

An ADT locator identifies an instance of the ADT specified by the locator specification. An array locator identifies an instance of the array type specified by the locator specification.

When the value at a site of data type large object string type, ADT or array is to be assigned to an embedded variable of the corresponding locator type, a locator is generated and assigned to the target.

A locator persists only until the current transaction is terminated, unless is has been held and has not subsequently been freed. A held locator persists until either the transaction is terminated with rollback, or until the termination of the SQL-session that created it.

4.5 Sites and operations on them

4.5.1 Sites

A *site* is a place that can hold 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 4.6.7.3, “Table constraints”).

4.5.2 Assignment and mutation

The instance at a site can be changed in two ways: by assignment or by mutation.

The operation of *assignment* replaces the instance at a site (known as the *target*) with a new instance of a (possibly, though not necessarily, different) value (known as the *source* value).

The term *mutation* is used to refer to an operation that changes the value of some attribute of an instance at a site whose data type is an abstract data type.

Neither assignment nor mutation has any 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

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 (ADTs, distinct types, and named row 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 of ADTs, fields 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*) together with a schema for collecting characters into strings (form-of-use), 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 in the Information Schema.

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.

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 many-to-many.

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.

A domain constraint does not apply to any site other than columns based on the domain.

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

4.6.4 Abstract data types and their components

4.6.4.1 Abstract data types

An *abstract data type* (known as an *ADT*) is a named, user-defined data type. A value of an ADT comprises a number of *attribute values*. Each attribute of an ADT has a data type, specified by an *attribute type* that is included in the descriptor of the ADT. 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 function known as an *observer function* that returns that value; an attribute value can be changed only by invoking a function known as a *mutator function*. An ADT instance can also be accessed by a *locator*.

An ADT may be defined to be a *subtype* of one or more other ADTs, known as *supertypes*. A subtype *inherits* every attribute of each of its supertypes, and may have additional attributes of its own. A value of a subtype may appear anywhere a value of any of its supertypes is allowed (this concept is known as *substitutability*).

4.6.4.2 Attributes

An *attribute* is a named component of an ADT descriptor. It has a data type, a default value, and a nullability characteristic.

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.

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 Named row types

A *named row type* is a user-defined data type, identical to a row type that can be specified by use of the row type constructor specified in ISO/IEC 9075-2. One or more base tables can be created, based on a named row type. A named row type may be a subtype of one or more other named row types.

4.6.7 Base tables and their components

4.6.7.1 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 named row type, it may be a *subtable* of one or more other base tables that are its *supertables*.

4.6.7.2 Columns

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

4.6.7.3 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. In addition, if the unique constraint was defined with PRIMARY KEY, then it requires that none of the values in the specified column or columns be the null value.

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).

4.6.7.4 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 action 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 an atomic compound statement.

4.6.8 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”, above.

4.6.9 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).

4.6.10 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", below.

4.6.11 Schema routines

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

4.6.12 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.

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.

A user who creates a schema object is automatically granted all possible privileges on it, with grant option.

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

4.6.13 Roles

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

A *role authorization* authorizes a grantee (see above) 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

For various reasons, the result of evaluating an expression 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 is implementation-dependent.

4.8 Communicating between the SQL-agent and the 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, for obvious reasons.

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

— The SQL-client module binding style. In this binding style, the user, using an implementation-defined mechanism, specifies a module to be used as an SQL-client module.

— The CLI 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 9075-5 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

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 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.

4.8.2.4 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 it 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 which can be 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 (it will not change any persistent SQL-data) or read-write.

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 will be executed completely or not at all, and that no update will be lost. The 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 module, each of which has certain properties, and contains various kinds of module object (also known as module contents). The principal module objects are one or more routines (see 4.10).

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 session.

4.10 Routines

4.10.1 General

The following table shows the terms used for the various possible combinations of SQL/External, SQL-invoked/Externally invoked, 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.

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.

4.10.2 Identity functions

If an SQL-invoked functions has a parameter 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 an *identity function*. The result of such a function is the value of the result parameter, possibly mutated.

Every mutator function is an identity 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 which case each is prepared as a (single SQL-statement) externally-invoked procedure when the SQL-client module is created, and executed when that procedure is called.

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. This Subclause will not mention those statements again, as their classification is entirely implementation-defined.

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 session with one SQL-server to a 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

ISO/IEC 9075-1, 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, and ISO/IEC 9075-7, Temporal, specify significant additions to SQL itself, in the first case by making SQL computationally complete, in the other two by specifying additional data types and operations on them.

ISO/IEC 9075-6, XA Specialization, specifies entry points by which an SQL-implementation may be invoked by a transaction manager.

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

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

ISO/IEC 9075-1 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.

— All locator types.

— Row types.

— Array types.

— Domains, abstract data types, and distinct types.

— 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 session with one SQL-server to a 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, possibly among many, SQL-environments. 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.

5.7 ISO/IEC 9075-6: XA Specialization (SQL/Transaction)

ISO/IEC 9075-6 is currently in preparation.

ISO/IEC 9075-6 provides SQL details for the XA interface.

5.8 ISO/IEC 9075-7: Temporal (SQL/Temporal)

ISO/IEC 9075-7 is currently in preparation.

ISO/IEC 9075-7 defines facilities to simplify the management of temporal data.

A period type is a data type each value of which represents a series of consecutive values of one of certain ordered data types — in particular, dates, times and timestamps.

As well as various operators on periods, ISO/IEC 9075-7 specifies aggregate operators on sets of periods and on tables with period-valued columns, for use in queries, constraints and database maintenance.

NOTE 3 – ISO/IEC 9075-7 uses the term “period” rather than “interval” because the latter is used with a different meaning in ISO/IEC 9075-2.

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.
!!	Introduces normal English text. This is used when the definition of a syntactic element is not expressed in BNF.

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

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

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

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

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

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

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

5) Apply steps 1 through 5 to any 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 4.29.1, "Status parameters", 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>.

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* of <C> 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 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 rules specified with “Case” conventions include a list of conditional sub-rules using “If” conventions. The first such “If” sub-rule whose condition is true is the effective sub-rule of the “Case” rule. The last sub-rule of a “Case” rule may specify “Otherwise”, in which case it is the effective sub-rule of the “Case” rule if no preceding “If” sub-rule 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 executing an SQL-statement* if *S1* is the SQL-statement contained in an <externally-invoked procedure> or an <SQL-invoked routine> that has been executed.

An <SQL procedure statement> *S1* is said to be executed as an *indirect result of executing an SQL-statement* if *S1* is a <triggered SQL statement> that is contained in some <trigger definition> and a triggering <SQL procedure statement> is executed.

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 UDT 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 instance of an object of a specified type. The concept of descriptor is used in specifying the semantics of SQL. It is not necessary that any descriptor exist in any particular form in any SQL-environment.

Some SQL objects cannot exist except in the context of other SQL objects. For example, columns cannot exist except in 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.

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 ISO/IEC 9075-1 and ISO/IEC 9075-2 depends 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, it supplements the Clause, Subclause, or Annex, respectively, in ISO/IEC 9075, 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.

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 it provides language specification particular to that part. A Subclause that is part of a Clause or Subclause identified as new is inherently new 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 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.

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, or replaces a Format item whose definition appears in ISO/IEC 9075-2, or defines a new Format item that does not have a definition at all in ISO/IEC 9075-2. Those Format items in the incremental part that modify a Format item whose definition appears in ISO/IEC 9075-2 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 have BNF nonterminal symbols identical to BNF nonterminal symbols of Format items in ISO/IEC 9075-2, but do not state that they include any alternatives from ISO/IEC 9075-2.

New Format items that have no correspondence to any Format item in ISO/IEC 9075-2 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.

In such indications, “SR” is used to mean “Syntax Rule”, “AR” is used to mean “Access Rule”, and “GR” is used to mean “General 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, then the table supplements the table in ISO/IEC 9075-2, 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.

The rows in modified tables are generally new rows to be effectively inserted into the corresponding table in ISO/IEC 9075-2, though in rare cases a row already in a table in ISO/IEC 9075-2 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 that identifies the characteristics of an SQL-implementation. Each part of ISO/IEC 9075 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> | <199x>

<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>

<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>

<199x> ::= 3| edition199x <left paren> 3 <right paren>

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

<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
```

NOTE 6 – For $n \geq 3$, <Part *n* yes> is specified in Part *n*.

Syntax Rules

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

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

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

      Case:

      i) <low>, then Entry SQL level.

      ii) <intermediate>, then Intermediate SQL level.

      iii) <high>, then Full SQL level.

   b) Otherwise:

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

      ii) <intermediate>, then level 2.

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

4) <parts> shall not be specified unless <SQL edition> is <199*x*>.

5) 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 content of each annex repeats what is stated elsewhere in the normative text.

7.1 Implementation-defined elements

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, 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

ISO/IEC 9075-2 and every incremental part 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 specified in ISO/IEC 9075-2.

— 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 additional support the requirements of one or more packages of features of ISO/IEC 9075, as specified in Annex C, "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 nonconforming 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. In such cases:

— It shall provide a flagger that identifies every SQL-statement that may produce such results.

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

An SQL-implementation shall provide a flagger with the following properties:

— Syntax only: the flagger shall identify every SQL-statement that is non-conforming without reference to schema contents

— Catalog Lookup: the flagger shall identify every SQL-statement that is non-conforming taking account of schema contents.

8.1.4 Claims of conformance

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

— A list of those parts to which conformance is claimed.

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

NOTE 8 – 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.

Annex A
(informative)

Maintenance and interpretation of SQL

ISO/IEC JTC1 provides formal procedures for revision, maintenance, and interpretation of JTC1 Standards. Section 6.13 of the JTC1 Directives, "Maintenance/correction of defects in JTC1 Standards", specifies procedures for creating and processing "defect reports". Defect reports may result in technical corrigenda, amendments, interpretations, or other commentary on an existing International Standard. In addition, SC21, the JTC1 subcommittee that developed ISO/IEC 9075, provides procedures for raising new "questions" about topics related to existing SC21 projects. Questions may result in interpretations, new project proposals, or possibly new defect reports.

Since publication of ISO/IEC 9075:1992, the following SC21 questions have resulted in formal interpretations of Database Language SQL. The first number in parentheses identifies the SC21 document in which the question was first raised, and the second number identifies the SC21 document in which the proposed interpretation was formally adopted.

1) to be provided

Since publication of ISO/IEC 9075:1992, several new defects have been discovered in the SQL language, leading to creation of the following defect reports.

1) to be provided

The SQL language corrections proposed in each defect report were accepted by SC21/WG3 in date to be provided (see SC21 N to be provided, city to be provided WG3 Resolutions). Further processing within SC21 was superseded by adoption of ISO/IEC 9075:199*x* as a replacement standard for ISO/IEC 9075:1992. All corrections to SQL proposed by these defect reports are included in ISO/IEC 9075.

Potential new questions or new defect reports addressing the specifications of ISO/IEC 9075 should be communicated to:

Secretariat, ISO/IEC JTC1/SC21/WG3
Standards Council of Canada
Ottawa, Ontario
Canada.

Annex B
(informative)

SQL Feature Taxonomy

This Annex describes a taxonomy of features of the SQL language. This Annex will not appear in the DIS version of SQL/Framework.

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

This Annex currently contains two tables. The first contains a taxonomy of SQL/Foundation features and should, as a result of the FCD ballot process, be moved to Part 2; the second contains a taxonomy of SQL/Bindings features and should, as a result of the FCD ballot process, be moved to Part 5.

The taxonomies of features for SQL/CLI (Part 3) and SQL/PSM (Part 4) appear in their respective documents (though without much detail to date).

Table 2, "SQL/Foundation feature taxonomy", contains a taxonomy of the features of the SQL language that are specified in ISO/IEC 9075-2. In this table, the first column contains a counter that may be used to quickly locate rows of the table; these values otherwise have no use and are not stable — that is, they are subject to change in future editions of or even Technical Corrigenda to ISO/IEC 9075 without notice.

The second column, "Feature ID", specifies the formal identification of each feature and each subfeature contained in the table. The Feature ID is stable and can be depended on to remain constant. A Feature ID value comprises either a letter and three digits or a letter, three digits, a hyphen, and one or two additional digits. Feature ID values containing a hyphen and additional digits indicate "subfeatures" that help to define complete features, which are in turn indicated by Feature ID values without a hyphen. Only entire features are used to specify the contents of Core SQL and various packages.

The "Feature Description" column contains a brief description of the feature or subfeature associated with the Feature ID value.

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

The Feature Description column must be enhanced during the FCD ballot period to identify specific BNF non-terminal symbols and specific Syntax and/or General Rules.

The final column, named "Core SQL?", provides the definition of the minimal conformance possibility for ISO/IEC 9075, called Core SQL. Features that are included in the definition of Core SQL contain the value "YES" in this column; their subfeatures contain the value "(yes)" for consistency. Features and subfeatures that are not part of Core SQL contain a dash ("—") in this column.

Table 2—SQL/Foundation feature taxonomy

	Feature ID	Feature Description	Core SQL?
1	E011	Numeric data types	YES
2	E011-1	The INTEGER and SMALLINT data types, integer literals, integer expressions, integer comparison, and integer assignment.	(yes)
3	E011-2	The REAL, DOUBLE PRECISON, and FLOAT data types, approximate numeric literals, approximate numeric expressions, approximate numeric comparison, and approximate numeric assignment.	(yes)
4	E011-3	The DECIMAL and NUMERIC data types, decimal & numeric literals, decimal & numeric expressions, decimal & numeric comparison, and decimal & numeric assignment.	(yes)
5	E011-4	The +, -, *, and / arithmetic operators	(yes)
6	E011-5	The =, <>, >, >=, <, and <= operators	(yes)
7	E011-6	Implicit casting among the numeric data types	(yes)
8	E021	Character data types	YES
9	E021-1	CHARACTER data type (including all its spellings)	(yes)
10	E021-2	CHARACTER VARYING data type (including all its spellings and implicit conversion to and from CHARACTER type)	(yes)
11	E021-3	Character literals, character comparison, and character assignment	(yes)
12	E021-4	The CHARACTER_LENGTH function	(yes)
13	E021-5	The OCTET_LENGTH function	(yes)
14	E021-6	The SUBSTRING function for use with CHARACTER and CHARACTER VARYING data types.	(yes)
15	E021-7	<character value expression>s by use of the concatenation operation on CHARACTER and CHARACTER VARYING data types	(yes)
16	E021-8	The UPPER and LOWER functions	(yes)
17	E021-9	TRIM function	(yes)
18	E021-10	Implicit casting among the character data types	(yes)
19	E031	Identifiers	YES
20	E031-1	Delimited identifiers	(yes)
21	E031-2	Lower case identifiers	(yes)
22	E031-3	Trailing underscore	(yes)
23	E041	Basic schema definition	YES
24	E041-1	<schema definition> as a means of defining base tables, and views together with the ability to grant permissions on those base tables and views. Note: Although schema definition must be supported, it need not be supported as an SQL statement.	(yes)
25	E041-2	<table definition> for persistent base tables	(yes)

Table 2—SQL/Foundation feature taxonomy (Cont.)

	Feature ID	Feature Description	Core SQL?
26	E041-3	<view definition>	(yes)
27	E041-4	WITH CHECK OPTION clause on <view definition> Note: See item F751 for the LOCAL and CASCADED options on WITH CHECK OPTION	(yes)
28	E041-5	<grant statement>	(yes)
29	E041-6	Allow the optional keyword TABLE on a <grant statement>	(yes)
30	E051	Basic query specification	YES
31	E051-1	SELECT DISTINCT	(yes)
32	E051-2	GROUP BY clause supported	(yes)
33	E051-3	A GROUP BY clause need not contain all the non-aggregated columns in the select list.	(yes)
34	E051-4	A GROUP BY clause can contain columns that are not in the select list.	(yes)
35	E051-5	<select list> items can be renamed (optional "AS <column name>" in <select sublist>)	(yes)
36	E051-6	HAVING clause supported	(yes)
37	E051-7	Qualified * in select list (<select list> item of the form "<table name>.*" or "<correlation name>.*")	(yes)
38	E051-8	<correlation names> can be specified in the FROM clause and can be used elsewhere in the <query specification> to distinguish among columns. (See also E061-13)	(yes)
39	E051-9	Support for the ability to rename the columns in the FROM clause (that is "FROM <table name> [[AS] <correlation name>] [<column name> {, <column name>}...]")	(yes)
40	E051-10	Derived tables supported in the FROM clause	(yes)
41	E051-11	Allow the optional keyword AS before a <correlation name>	(yes)
42	E061	Basic predicates and search conditions	YES
43	E061-1	<comparison predicate>	(yes)
44	E061-2	<between predicate>	(yes)
45	E061-3	<in predicate>	(yes)
46	E061-4	<like predicate>	(yes)
47	E061-5	<like escape clause>	(yes)
48	E061-6	<null predicate>	(yes)
49	E061-7	<quantified comparison predicate>	(yes)
50	E061-8	<exists predicate>	(yes)
51	E061-9	Subqueries in <comparison predicate>	(yes)
52	E061-10	Subqueries in <exists predicate>	(yes)

Table 2—SQL/Foundation feature taxonomy (Cont.)

	Feature ID	Feature Description	Core SQL?
53	E061-11	Subqueries in <in predicate>	(yes)
54	E061-12	Subqueries in <quantified predicate>	(yes)
55	E061-13	Correlated subqueries (<correlation names> used in a sub-query as correlated reference to a column in the outer query)	(yes)
56	E061-14	<search condition> (Two or more predicates combined using the AND, OR, and NOT logical operators)	(yes)
57	E071	Basic query expressions	YES
58	E071-1	UNION table operator	(yes)
59	E071-2	UNION ALL table operator	(yes)
60	E071-3	EXCEPT table operator	(yes)
61	E071-4	EXCEPT ALL table operator	(yes)
62	E071-5	Columns combined via table operators need not have exactly the same data type.	(yes)
63	E071-6	Support of table operators within a subquery	(yes)
64	E081	Basic Privileges	YES
65	E081-1	SELECT privilege	(yes)
66	E081-2	DELETE privilege	(yes)
67	E081-3	INSERT privilege at the table level	(yes)
68	E081-4	UPDATE privilege at the table level	(yes)
69	E081-5	UPDATE privilege at the column level	(yes)
70	E081-6	REFERENCES privilege at the table level	(yes)
71	E081-7	REFERENCES privilege at the column level	(yes)
72	E081-8	WITH GRANT OPTION	(yes)
73	E091	Set functions	YES
74	E091-1	AVG	(yes)
75	E091-2	COUNT	(yes)
76	E091-3	MAX	(yes)
77	E091-4	MIN	(yes)
78	E091-5	SUM	(yes)
79	E091-6	ALL quantifier	(yes)
80	E091-7	DISTINCT quantifier	(yes)
81	E101	Basic data manipulation	YES
82	E101-1	<insert statement>	(yes)
83	E101-2	The VALUES clause in an <insert statement> used to insert multiple rows with one invocation.	YES

Table 2—SQL/Foundation feature taxonomy (Cont.)

	Feature ID	Feature Description	Core SQL?
84	E101-3	<update statement: searched>	(yes)
85	E101-4	<delete statement: searched>	(yes)
86	E111	<select statement: single row>	YES
87	E121	Basic cursor support	YES
88	E121-1	<declare cursor>	(yes)
89	E121-2	Columns in the <order by clause> need not also be specified in the <select list>	(yes)
90	E121-3	Value expressions in ORDER BY clause (that is, a <sort key> element is not restricted to being either a <column name> or an <integer> that designates a column>)	(yes)
91	E121-4	<open statement>	(yes)
92	E121-5	<fetch statement>	(yes)
93	E121-6	<update statement: positioned>	(yes)
94	E121-7	<delete statement: positioned>	(yes)
95	E121-8	<close statement>	(yes)
96	E121-9	Read-only scrollable cursor support	(yes)
97	E121-10	FETCH NEXT	(yes)
98	E121-11	FETCH FIRST	(yes)
99	E121-12	FETCH LAST	(yes)
100	E121-13	FETCH PRIOR	(yes)
101	E121-14	FETCH ABSOLUTE	(yes)
102	E121-15	FETCH RELATIVE	(yes)
103	E121-16	Support the optional FROM clause in <fetch statement>	(yes)
104	E131	Null value support (nulls in lieu of values)	YES
105	E141	Basic integrity constraints	YES
106	E141-1	NOT NULL constraints	(yes)
107	E141-2	UNIQUE constraints	(yes)
108	E141-3	PRIMARY KEY constraints	(yes)
109	E141-4	Basic FOREIGN KEY constraint with the NO ACTION default for both referential delete action and referential update action.	(yes)
110	E141-5	Referential delete actions	(yes)
111	E141-6	CHECK constraints	(yes)
112	E141-7	Column defaults	(yes)
113	E141-8	NOT NULL inferred on UNIQUE and PRIMARY KEY	(yes)
114	E141-9	Named constraints	(yes)

Table 2—SQL/Foundation feature taxonomy (Cont.)

	Feature ID	Feature Description	Core SQL?
115	E141-10	Referential name order (names in a foreign key can be specified in any order)	(yes)
116	E151	Transaction support	YES
117	E151-1	<commit statement>	(yes)
118	E151-2	<rollback statement>	(yes)
119	E151-3	Support for the ISOLATION LEVEL clause on the SET TRANSACTION statement. (Note: This does not mean that the semantics of the four isolation levels must be supported. An implementation may simply support the syntax and to escalate any specified level to a higher level.)	(yes)
120	E151-4	Support for the READ ONLY and READ WRITE clauses on the SET TRANSACTION statement	(yes)
121	E151-5	Support for the READ ONLY and UPDATE clauses on the DECLARE CURSOR statement.	(yes)
122	E151-6	A <query expression> is updateable even though its <where clause> contains a <subquery>.	(yes)
123	E151-7	Allow the word WORK not to be specified.	(yes)
124	E161	SQL comments	YES
125	E161-1	Leading double <minus sign> comments	(yes)
126	E161-2	Bracketed comments (/*...*/ comments)	(yes)
127	E171	SQLSTATE support	YES
128	E182	Module language	YES
129	E19	Basic flagging	YES
130	F021	Basic information schema (Support of the COLUMNS, TABLES, and VIEWS views in the INFORMATION_SCHEMA)	YES
131	F031	Basic schema manipulation	YES
132	F031-1	CREATE TABLE statement to create persistent base tables	(yes)
133	F031-2	CREATE VIEW statement	(yes)
134	F031-3	GRANT statement	(yes)
135	F031-4	ALTER TABLE statement, ADD COLUMN clause	(yes)
136	F031-5	ALTER TABLE statement, DROP COLUMN clause	(yes)
137	F031-6	ALTER TABLE statement, DROP COLUMN CASCADE clause	(yes)
138	F031-7	ALTER TABLE statement, DROP COLUMN RESTRICT clause	(yes)
139	F031-8	ALTER TABLE statement, ALTER COLUMN clause	(yes)
140	F031-9	ALTER TABLE statement, ADD CONSTRAINT clause	(yes)
141	F031-10	ALTER TABLE statement, DROP CONSTRAINT clause	(yes)
142	F031-11	DROP TABLE statement	(yes)

Table 2—SQL/Foundation feature taxonomy (Cont.)

	Feature ID	Feature Description	Core SQL?
143	F031-12	DROP TABLE statement, CASCADE clause	(yes)
144	F031-13	DROP TABLE statement, RESTRICT clause	(yes)
145	F031-14	DROP VIEW statement	(yes)
146	F031-15	DROP VIEW statement, CASCADE clause	(yes)
147	F031-16	DROP VIEW statement, RESTRICT clause	(yes)
148	F031-17	REVOKE statement	(yes)
149	F031-18	REVOKE statement, CASCADE clause	(yes)
150	F031-19	REVOKE statement, RESTRICT clause	(yes)
151	F041	Basic joined table	YES
152	F041-1	Inner join (but not necessarily the INNER keyword)	(yes)
153	F041-2	INNER keyword	(yes)
154	F041-3	Left Outer Join	(yes)
155	F041-4	Right Outer Join	(yes)
156	F041-5	Outer joins can be nested	(yes)
157	F041-6	Column names in ON clause can be in different order than those in the OUTER JOIN clause	(yes)
158	F041-7	The inner table in a left or right outer join can also be used in an inner join	(yes)
159	F041-8	All comparison operators are supported (rather than just =)	(yes)
160	F051	Basic date & time	YES
161	F051-1	DATE data type (including support of DATE literal)	(yes)
162	F051-2	TIME data type (including support of TIME literal) with fractional seconds precision of at least 0.	(yes)
163	F051-3	TIMESTAMP data type (including support of TIMESTAMP literal) with fractional seconds precision of at least 0 and 6.	(yes)
164	F051-4	Comparison predicate on like date & time data types	(yes)
165	F051-5	Explicit CAST between datetime types and CHARACTER & CHARACTER VARYING	(yes)
166	F051-6	CURRENT_DATE	(yes)
167	F051-7	CURRENT_TIME	(yes)
168	F051-8	CURRENT_TIMESTAMP	(yes)
169	F052	Interval data type	—
170	F081	UNION in views	YES
171	F121	Get diagnostics	—
172	F131	Grouped operations	YES

Table 2—SQL/Foundation feature taxonomy (Cont.)

	Feature ID	Feature Description	Core SQL?
173	F131-1	Even though a table in the FROM clause is a grouped view, the query can contain a WHERE, GROUP BY, or HAVING	(yes)
174	F131-2	Even though a table in the FROM clause is a grouped view, multiple tables can be specified in the query	(yes)
175	F131-3	Even though a table in the FROM clause is a grouped view, the select list can contain a <set function>.	(yes)
176	F131-4	A subquery within a comparison predicate cannot contain a GROUP BY clause or a HAVING clause and can identify a grouped view.	(yes)
177	F131-5	The table in the FROM clause of a single row SELECT statement can be a grouped view. Also a single row SELECT statement may specify a GROUP BY clause or HAVING clause.	(yes)
178	F171	Multiple schemas per user	YES
179	F181	Multiple module support (the ability to associate multiple host compilation units with a single SQL-session at one time)	YES
180	F201	CAST functions (excluding support for casting the INTERVAL data type)	YES
181	F221	Explicit defaults	YES
182	F222	DEFAULT VALUES support in an <insert statement>	—
183	F231	Privilege Tables	YES
184	F231-1	TABLE_PRIVILEGES view	(yes)
185	F231-2	COLUMN_PRIVILEGES view	(yes)
186	F231-3	USAGE_PRIVILEGES view	(yes)
187	F251	Domain support	—
188	F261	CASE expression	YES
189	F261-1	<simple case>	(yes)
190	F261-2	<searched case>	(yes)
191	F261-3	NULLIF	(yes)
192	F261-4	COALESCE	(yes)
193	F271	Compound character literals	—
194	F281	LIKE enhancements	—
195	F291	UNIQUE predicate	—
196	F301	<corresponding specification> in <query expression>s	—
197	F302	INTERSECT table operator	—
198	F311	Schema definition statement	YES
199	F321	User authorization	—
200	F331	Constraint tables	YES
201	F331-1	TABLE_CONSTRAINTS view	(yes)

Table 2—SQL/Foundation feature taxonomy (Cont.)

	Feature ID	Feature Description	Core SQL?
202	F331-2	REFERENTIAL_CONSTRAINTS view	(yes)
203	F331-3	CHECK_CONSTRAINTS view	(yes)
204	F341	Usage tables	—
205	F361	User authorization	—
206	F381	Extended schema manipulation	—
207	F391	Long identifiers	—
208	F401	Full outer join	—
209	F401-1	Natural Join	—
210	F411	Time zone specification	—
211	F421	National character	—
212	F441	Extended set function support	—
213	F451	Character set definition	—
214	F461	Named character sets	—
215	F471	Scalar subquery values	YES
216	F481	Expanded NULL predicate	YES
217	F491	Constraint management	—
218	F501	Features and conformance tables	YES
219	F501-1	SQL_FEATURES	(yes)
220	F501-2	SQL_SIZING	(yes)
221	F511	BIT data type	—
222	F521	Assertions	—
223	F531	Temporary tables	—
224	F551	Full datetime	—
225	F561	Full value expressions	—
226	F571	Truth value tests	—
227	F581	The POSITION function for use with CHARACTER, CHARACTER VARYING, and LOB data types	—
228	F611	Indicator data types	—
229	F641	Row and table constructors	—
230	F651	Catalog name qualifiers	—
231	F661	Simple tables	—
232	F671	Subqueries in CHECK	—
233	F681	Union and cross join	—

Table 2—SQL/Foundation feature taxonomy (Cont.)

	Feature ID	Feature Description	Core SQL?
234	F691	Collation and translation	—
235	F701	Referential update actions	—
236	F721	Deferable constraints	—
237	F731	INSERT column privileges	—
238	F741	Referential MATCH types	—
239	F751	View CHECK enhancements	—
240	F761	Session management	—
241	F771	Connection management	—
242	F781	Self-referencing operations	—
243	F791	Insensitive cursors	—
244	F811	Extended flagging	—
245	F821	Local table references	—
246	F831	Full cursor update	—
247	T011	Timestamp in information schema for configuration management	—
248	T031	BOOLEAN data type	—
249	T041	Basic LOB data type support	YES
250	T041-1	BLOB data type	(yes)
251	T041-2	CLOB data type	(yes)
252	T041-3	LENGTH and SUBSTRING function support for LOB data types	(yes)
253	T041-4	concatenation of LOB data types	(yes)
254	T041-5	non-holdable locator for LOB data types	(yes)
255	T042	Extended LOB data type support	—
256	T042-1	OVERLAY function	—
257	T042-*n*	*other subfeatures to be specified*	—
258	T071	CASCADE option for DROP COLLATION	—
259	T121	WITH in <query expression>	—
260	T131	Recursive query	—
261	T141	SIMILAR predicate	—
262	T151	DISTINCT predicate	—
263	T161	Optional interval qualifier	—
264	T171	LIKE clause in table definition	—
265	T191	Referential action RESTRICT	—
266	T201	Comparable data types for referential constraints	YES

Table 2—SQL/Foundation feature taxonomy (Cont.)

	Feature ID	Feature Description	Core SQL?
267	T211	Triggers	YES
268	T211-1	Support for triggers activated on UPDATE, INSERT, DELETE of one base table.	(yes)
269	T211-2	Support for BEFORE triggers that are applied before any modifications are made to the database. These triggers have access to old (delete, update) and new (insert, update) rows.	(yes)
270	T211-3	Support for AFTER triggers that are applied before any modifications are made to the database. These triggers have access to both old (delete, update) and new (insert, update) rows and transition tables.	(yes)
271	T211-4	Support for triggers that are to be applied once for each row of the subject table that is affected by the triggering SQL operation.	(yes)
272	T211-5	Ability to specify a search condition that must be true before the trigger is invoked.	(yes)
273	T211-6	Support for run-time rules for the interaction of triggers and constraints.	(yes)
274	T211-7	TRIGGER privilege	(yes)
275	T211-8	Multiple triggers for the same the event are executed in the order in which they were created in the catalog.	(yes)
276	T212	Triggers applied once for the triggering statement	—
277	T221	WITH HOLD cursors	YES
278	T231	SENSITIVE cursors	—
279	T241	START TRANSACTION statement	—
280	T251	LOCAL option for SET TRANSACTION statement	—
281	T261	Chained transactions	—
282	T271	Savepoints	—
283	T281	SELECT privilege with column granularity	—
284	T291	Static and Dynamic execution rights	—
285	T301	Functional Dependencies	—
286	T321	SQL-invoked routines	YES
287	T321-1	User-defined functions	(yes)
288	T321-2	User-defined stored procedures	(yes)
289	T321-3	<routine invocation>	(yes)
290	T321-4	<call statement>	(yes)
291	T321-5	<return statement>	(yes)
292	T331	Roles	—
293	T361	User-defined aggregate operators	—
294	T371	Quantified predicate extensions	—

Table 2—SQL/Foundation feature taxonomy (Cont.)

	Feature ID	Feature Description	Core SQL?
295	T391	Table name not required in <delete statement: positioned> or <update statement: poitioned>	—
296	T401	INSERT into a cursor	—
297	T411	ROW may be specified in an UPDATE statement	—
298	T421	Character Sets *To do during the ballot: Give this a better name and, perhaps, merge it along with F451 and F461.*	—
299	T431	CUBE and ROLLUP operations	—
300	T441	ABS and MOD functions	—
301	T461	Symmetric <between predicate>	—
302	T471	Result sets return value	—
303	O011	Minimum user-defined data type support (distinct types)	YES
304	O021	Basic user-defined data types (Support for structured types — ADTs and named row types — with the exception of those features listed under Enhanced ADTs)	—
305	O022	Enhanced user-defined data types	—
306	O022-1	Constructor option	—
307	O022-2	Attribute default	—
308	O022-3	Multiple inheritance	—
309	O022-4	Public, private, protected specification on attributes (ANSI only)	—
310	O022-5	Ordering clause in type definition	—
311	O041	Reference types	—
312	O051	Create table of type	—
313	O061	ALTER TABLE <add named row type>	—
314	O071	SQL paths in function and type name resolution	—
315	O081	Subtables	—
316	O091	Basic array support	YES
317	O091-1	arrays of built-in data types	(yes)
318	O091-2	arrays of distinct types	(yes)
319	O092	Arrays of UDTs	—
320	O094	Arrays of reference types	—
321	O111	ONLY in query expressions (to restrict subtable search)	—
322	O121	Dereference operation (path expressions)	—
323	O131	Reference operation	—
324	O141	Attribute & field reference	—

Table 2—SQL/Foundation feature taxonomy (Cont.)

	Feature ID	Feature Description	Core SQL?
325	O141-1	Observer reference	—
326	O141-2	Field reference	—
327	O151	Type predicate	—
328	O161	<subtype treatment>	—
329	O171	Array expressions	YES
330	O191	Basic SQL routines on user-defined types (with dynamic dispatch)	—
331	O192	Basic SQL routines on user-defined types	—
332	O192-1	Identity functions	—
333	O192-2	Generalized expressions	—
334	O201	SQL routines on arrays	YES
335	O201-1	Array parameters	(yes)
336	O201-2	Array as result type of functions	(yes)
337	O211	User-defined cast functions	—
338	O231	ADT locators	—
339	O232	Array locators	YES

Table 3, "SQL/Bindings feature taxonomy", contains a taxonomy of the features of the SQL language that are specified in ISO/IEC 9075-5. In this table, the first column contains a counter that may be used to quickly locate rows of the table; these values otherwise have no use and are not stable — that is, they are subject to change in future editions of or even Technical Corrigenda to ISO/IEC 9075 without notice.

The second column, "Feature ID", specifies the formal identification of each feature and each subfeature contained in the table. The Feature ID is stable and can be depended on to remain constant. A Feature ID value comprises either a letter and three digits or a letter, three digits, a hyphen, and one or two additional digits. Feature ID values containing a hyphen and additional digits indicate "subfeatures" that help to define complete features, which are in turn indicated by Feature ID values without a hyphen. Only entire features are used to specify the contents of Core SQL and various packages.

The "Feature Description" column contains a brief description of the feature or subfeature associated with the Feature ID value.

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

The Feature Description column must be enhanced during the FCD ballot period to identify specific BNF non-terminal symbols and specific Syntax and/or General Rules.

The final column, named "Core SQL?", provides the definition of the minimal conformance possibility for ISO/IEC 9075, called Core SQL. Features that are included in the definition of Core SQL contain the value "YES" in this column; their subfeatures contain the value "(yes)" for consistency. Features and subfeatures that are not part of Core SQL contain a dash ("—") in this column.

Table 3—SQL/Bindings feature taxonomy

	Feature ID	Feature Description	Core SQL?
1	B001	Embedded SQL	YES
2	B001-1	Embedded Ada	(yes)
3	B001-2	Embedded C	(yes)
4	B001-3	Embedded COBOL	(yes)
5	B001-4	Embedded Fortran	(yes)
6	B001-5	Embedded MUMPS	(yes)
7	B001-6	Embedded Pascal	(yes)
8	B001-7	Embedded PL/I	(yes)
9	B002	Basic dynamic SQL (Support of the PREPARE, EXECUTE, EXECUTE IMMEDIATE, DESCRIBE, ALLOCATE DESCRIPTOR, and DEALLOCATE DESCRIPTOR statements in some host language)	—
10	B003	Extended dynamic SQL	—
11	B003-1	<describe input> statement	—
12	B003-*n*	*other subfeatures to be specified*	—
13	B004	Extensions to embedded SQL exception declarations	—

Annex C
(informative)

SQL Packages

This Annex describes *packages* of features of the SQL language. In addition to conformance claims to Core SQL, SQL-implementations may elect to claim conformance to one or more packages. While this Annex specifies several such packages, it is expected that packages may be specified elsewhere, outside of the scope of ISO/IEC 9075.

C.1 Enhanced datetime facilities

The package called "enhanced datetime facilities" comprises the following features of the SQL language as specified in the SQL Feature Taxonomy Annex of the various parts of ISO/IEC 9075.

— F052 (Interval data type)

— F411 (Time zone specification)

— F551 (Full datetime)

— T161 (Optional interval qualifier)

C.2 Enhanced integrity management

The package called "enhanced integrity management" comprises the following features of the SQL language as specified in the SQL Feature Taxonomy Annex of the various parts of ISO/IEC 9075.

— F521 (Assertions)

— F701 (Referential update actions)

— F491 (Constraint management)

— F671 (Subqueries in CHECK constraints)

— T212 (FOR EACH STATEMENT triggers)

— T191 (Referential action RESTRICT)

C.3 OLAP facilities

The package called "OLAP facilities" comprises the following features of the SQL language as specified in the SQL Feature Taxonomy Annex of the various parts of ISO/IEC 9075.

— T431 (CUBE and ROLLUP)

— F302 (INTERSECT table operator)

— F641 (Row and table constructors)

— F401 (FULL OUTER JOIN)

— F471 (Scalar subquery values)

C.4 PSM

The package called "PSM" comprises the following features of the SQL language as specified in the SQL Feature Taxonomy Annex of the various parts of ISO/IEC 9075.

— P01 (Stored modules (<SQL-server module definition>))

— P02 (Computational completeness)

— P03 (Information Schema views)

C.5 CLI

The package called "cli" comprises the following features of the SQL language as specified in the SQL Feature Taxonomy Annex of the various parts of ISO/IEC 9075.

— C01 (SQL/CLI)

C.6 Basic object support

The package called "basic object support" comprises the following features of the SQL language as specified in the SQL Feature Taxonomy Annex of the various parts of ISO/IEC 9075.

— O021 (Basic user-defined types, including single inheritance)

— O041 (Reference types)

— O051 (CREATE TABLE of type)

— O092 (Arrays of UDTs)

— O094 (Arrays of reference types)

— O121 (Dereference operation)

— O131 (Reference operation)

— O141 (Attribute & field reference)

— O191 (Basic SQL routines on user-defined types, including dynamic dispatch)

C.7 Enhanced object support

The package called “enhanced object support” comprises the following features of the SQL language as specified in the SQL Feature Taxonomy Annex of the various parts of ISO/IEC 9075.

— O022 (Enhanced user-defined types, including constructor option, attribute defaults, multiple inheritance, and ordering clause)

— O061 (ALTER TABLE, ADD named row type)

— O071 (SQL-paths in function and type name resolution)

— O081 (Subtables)

— O111 (ONLY in query expressions (to restrict subtable search))

— O151 (Type predicate)

— O161 (<subtype treatment>)

— O192 (SQL routines on user-defined types, including identity functions and generalized expressions)

— O211 (User-defined cast functions)

— O231 (ADT locators)

Index

Index entries appearing in **boldface** indicate the page where the word, phrase, or BNF nonterminal was defined; index entries appearing in *italics* indicate a page where the BNF nonterminal was used in a Format; and index entries appearing in roman type indicate a page where the word, phrase, or BNF nonterminal was used in a heading, Function, Syntax Rule, Access Rule, General Rule, Leveling Rule, Table, or other descriptive text.

<1987> • ***40***
<1989> • ***40***
<1992> • *40*, **41**

— A —

ABS • 62
ABSOLUTE • 55
abstract data type • 14, 16, 26
access mode • 21
activated • 61
active • 23, 26, 29
Ada • 64
ADD • 56, 67
ADMIN • 18
ADT • 13, 14, 16, 22, 62, 63, 67
AFTER • 17, 61
AFTER trigger • 61
ALL • 54
ALTER • 56, 62, 67
ALTER TABLE • 56
AND • 54
applicable • 45
application program • 19, 26, 28
approximate numeric • 11, 52
<arc1> • ***40***
<arc2> • ***40***
<arc3> • ***40***
array • 12, 13, 62, 63
array locator • 13
array type • 12, 13
AS • 53
assertion • 14, 18, 37
assignment • 14, 27, 33, 52
atomic • 5, 10, 11, 17, 21
attribute • 5, 6, 14, 16, 28, 37, 62, 67
Attribute • 16, 62, 67
ATTRIBUTES • 22, 51, 63
attribute type • 16
attribute values • 16
authorization identifier • 7, 8, 9, 18
AVG • 54

— B —

<1989 base> • ***40***
based • 3, 6, 11, 15, 16, 18, 25, 33, 37
base table • 7, 9, 10, 12, 13, 14, 16, 17, 52, 56, 61
base tables • 7, 9, 10, 12, 14, 16, 52, 56
BEFORE • 17, 61
BEFORE trigger • 61
be included in • 37
<between predicate> • 53, 62
binary large object • 11
binary string type • 11
binding style • 8, 19
binding styles • 19
bit string type • 11
BLOB • 11, 13, 60
BNF nonterminal symbol • 31, 38, 39
Boolean • 11, 25
BOOLEAN • 60
built-in function • 22
BY • 53, 55, 58

— C —

<call statement> • 61
candidate key • 26
candidate routine • 27
CASCADED • 53
CAST • 57, 58
catalog • 7, 9, 28, 45, 61
character • 5, 6, 11, 13, 14, 15, 16, 17, 19, 20, 22, 23, 28, 31, 32, 40, 52, 58, 59
CHARACTER • 52, 57, 59
character large object • 11, 13
character repertoire • 11, 15
character set • 15, 28, 59
character type • 11
<character value expression> • 52
CHARACTER_LENGTH • 52
CHECK • 53, 55, 59, 60, 65
check constraint • 17, 18
CHECK_CONSTRAINTS • 59
CLOB • 11, 13, 60
<close statement> • 55

cluster • 9
COALESCE • 58
COBOL • 64
collating sequence • 15
collation • 15
collection • 6, 9, 10, 12, 18, 36, 37, 46
column • 6, 10, 14, 15, 17, 18, 19, 29, 36, 37, 40, 51, 53, 54, 55, 60, 61, 63
<column name> • 53, 55
COLUMNS • 56
COLUMN_PRIVILEGES • 58
COMMIT • 21
<commit statement> • 56
COMMITTED • 21
<comparison predicate> • 53
compilation unit • 5, 7, 22, 28, 46, 58
component • 5, 6, 7, 14, 15, 16, 17, 22
conditions • 21, 27, 28, 34, 35, 53
conformance • 6, 32, 40, 41, 42, 45, 46, 47, 51, 59, 63, 65
Conformance • 32, 33, 34, 40, 45
conforming SQL-implementation • 5, 6, 19, 21
Connect • 60
constraint mode • 18, 19
CONSTRAINTS • 58, 59
contain • 3, 7, 8, 9, 10, 14, 19, 21, 22, 25, 27, 28, 29, 33, 34, 35, 36, 43, 46, 51, 53, 56, 58, 63
contained in • 34, 36, 51, 63
containing • 7, 21, 22, 34, 35, 36, 51, 63
contains • 7, 8, 9, 10, 19, 21, 25, 28, 34, 35, 43, 46, 51, 56, 63
<correlation name> • 53
COUNT • 54
covered • 5, 49
CREATE • 56, 66
created base table • 10
CUBE • 62, 66
CURRENT_DATE • 57
CURRENT_TIMESTAMP • 57

— D —

Data • 1, 3, 20, 25, 40, 49
database • 7, 28, 29, 61
data type • 5, 6, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 22, 25, 26, 29, 52, 54, 57, 58, 59, 60, 62, 65
date • 10, 12, 19, 21, 23, 26, 27, 29, 43, 49, 51, 55, 56, 57, 59, 60, 61, 62, 65
DATE • 57
datetime types • 12, 57
DAY • 12
day-time intervals • 12
DEALLOCATE • 64
DECIMAL • 52
DECLARE • 56
<declare cursor> • 55
declared temporary table • 10
DEFAULT • 58
default collation • 15
deferrable • 18, 19
deferred • 19
definition schema • 9
Definition Schema • 9
DEFINITION_SCHEMA • 9
degree • 13, 21
DELETE • 17, 18, 54, 61
<delete statement: positioned> • 55, 62
<delete statement: searched> • 55
depend • 5, 6, 10, 13, 14, 17, 19, 26, 27, 33, 34, 35, 37, 38, 43, 51, 63
dependent • 5, 6, 13, 14, 19, 27, 33, 34, 35, 37, 43
depends on • 17, 37, 38
derived table • 10
DESCRIBE • 64
Description • 33, 39, 51, 52, 63, 64
descriptor • 5, 6, 9, 10, 14, 16, 20, 26, 27, 28, 29, 33, 37
DESCRIPTOR • 64
descriptor area • 20, 28, 29, 33
deterministic • 19, 35
diagnostics area • 8, 21, 24, 26
directly • 16, 20, 23, 28, 29
direct result of executing an SQL-statement • 36
distinct • 6, 10, 12, 14, 15, 16, 17, 26, 62
DISTINCT • 53, 54, 60
distinct type • 14, 16, 26, 62
domain • 14, 15, 18, 37
domain constraint • 14, 15, 18
DOUBLE • 52
DROP • 56, 57, 60
duplicate • 19

— E —

EACH • 65
Editor's Note • 51, 63
effective • 35, 37, 40
effectively • 35, 37, 40
element • 6, 12, 31, 32, 33, 35, 36, 43, 46, 55
element type • 12
embedded • 13, 23, 28, 29, 64
encapsulated • 16
encapsulation • 16
environment • 6, 7, 9, 21, 22, 25, 27, 37, 40
equivalent • 26, 28, 36, 37, 40
exact numeric • 11
exception • 20, 24, 27, 29, 33, 34, 35, 62, 64
Execute • 8
EXECUTE • 18, 64
<exists predicate> • 53
explicit • 8, 14, 31, 33, 36, 37, 38, 46
Explicit • 57, 58
<externally-invoked procedure> • 36
externally-invoked routine • 22
external routine • 22
EXTRACT • 10

— F —

FETCH • 55
<fetch statement> • 55
FIRST • 55
FLOAT • 52
FOREIGN • 55
form-of-use • 15
FROM • 53, 55, 58
function • 13, 15, 16, 22, 23, 25, 26, 29, 31, 35, 45, 52, 54, 58, 59, 60, 61, 62, 63, 67
functional dependencies • 26

— G —

generally contain • 34
global temporary table • 10
GRANT • 54, 56
<grant statement> • 53
GROUP • 53, 58
grouped view • 58

— H —

handle • 27
HAVING • 53, 58
<high> • ***41***
HOLD • 61
host language • 19, 20, 28, 29, 45, 64
HOUR • 12

— I —

identity function • 22, 67
immediate • 19, 20, 29, 31, 34, 36, 38, 39, 46
IMMEDIATE • 64
immediately contain • 34, 36
implementation • 1, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 19, 20, 21, 22, 23, 25, 26, 27, 28, 33, 34, 35, 37, 40, 43, 45, 46, 47, 56, 65
implementation-defined • 5, 7, 8, 9, 11, 19, 22, 23, 28, 35, 46, 47
implementation-dependent • 5, 14, 19, 33, 34, 35
implicit • 21, 36, 52
Implicit • 52
include • 5, 7, 9, 11, 12, 16, 27, 28, 35, 37, 39, 46, 49, 51, 63
incremental parts • 38, 47
independent • 5, 13, 27, 37
Index typography • 40
indicator • 20
Indicator • 20, 59
indicator parameter • 20
indirect result of executing an SQL-statement • 36
inessential • 35
Information Schema • 9, 15, 18, 27, 33, 40, 45, 66
INFORMATION_SCHEMA • 9, 56
inherit • 14, 16, 62, 66, 67
inherits • 16
INNER • 57
innermost • 31, 34
inner table • 57
<in predicate> • 53, 54
INSERT • 17, 18, 54, 60, 61, 62
<insert statement> • 54, 58
instance • 5, 6, 7, 13, 14, 16, 31, 32, 33, 34, 36, 37, 39
INTEGER • 52
integrity constraint • 17, 55
<integrity no> • ***41***
<integrity yes> • ***41***
interface • 3, 19, 25, 29
<intermediate> • ***41***
INTERSECT • 58, 66
INTERVAL • 58
interval type • 12, 26
intervening • 34, 38
is a part of • 36, 37
is dependent on • 37
ISOLATION • 56
isolation level • 21, 56

— J —

JOIN • 57

— K —

KEY • 17, 55
known not nullable • 14

— L —

LAST • 55
<left paren> • *40*, *41*
LENGTH • 52, 60
<level> • ***41***
level of conformance • 46
LIKE • 58, 60
<like predicate> • 53
LOB • 59, 60
LOCAL • 53, 61
local temporary table • 10
locator • 13, 16, 26, 60, 63, 67
Locator • 13
locator type • 13, 26
LOW • 52
<low> • ***41***
LOWER • 52

— M —

MATCH • 60
match type • 17
MAX • 54
MIN • 54
<minus sign> • 56
MINUTE • 12
MOD • 62
<modified item> • ***38***
module • 3, 7, 8, 10, 14, 18, 19, 20, 21, 22, 23, 27, 45, 46, 58, 66
MONTH • 12
MUMPS • 64
mutation • 14
mutator function • 16, 22

— N —

Name • 16, 55, 59
named row type • 12, 16, 62, 67
NATIONAL • 11
national character repertoire • 11
NEXT • 55
NO • 55
non-deterministic • 19
NOT • 54, 55
not deferrable • 18
null • 5, 11, 14, 16, 17, 20, 53, 55
NULL • 11, 55, 59
nullability characteristic • 14, 16, 17
NULLIF • 58
<null predicate> • 53
null value • 5, 11, 14, 17, 20
NUMERIC • 52

— O —

object • 5, 6, 9, 10, 11, 13, 14, 15, 17, 18, 21, 22, 23, 26, 32, 33, 37, 40, 41, 45, 66, 67
object identifier • 40
observer function • 16
octet • 11
OCTET_LENGTH • 52
ON • 57
ONLY • 56, 62, 67
<open statement> • 55
operator • 29, 31, 35, 52, 54, 57, 58, 61, 66
Option • 60, 65
OPTION • 18, 53, 54
OR • 54
<order by clause> • 55
OUTER • 57, 66
outermost • 34
output parameter • 20
OVERLAY • 60
overloaded • 22
owned by • 9

— P —

<1989 package> • *40*, **41**
parameter • 10, 13, 16, 20, 21, 22, 23, 27, 33, 36, 63
Part 2 • 3, 51
Part 3 • 3, 51
Part 4 • 3, 51
Part 5 • 3, 51
Part 6 • 3
Part 7 • 3
<Part *n*> • **41**
<Part *n* no> • ***41***
<Part *n* yes> • ***41***
part of • 6, 8, 10, 15, 25, 34, 35, 36, 37, 38, 40, 43, 45, 46, 51, 63
<parts> • ***41***, 42
Pascal • 64
period • 12, 29, 51, 63
period type • 29
permitted • 8, 11, 12, 19, 43
persist • 1, 5, 9, 10, 13, 14, 18, 21, 23, 52, 56
persistent • 1, 5, 9, 10, 13, 14, 18, 21, 23, 52, 56
persistent base table • 10, 52
POSITION • 59
possibly modify SQL-data • 35
possibly non-deterministic • 19
possibly nullable • 14
precede • 33, 35
precision • 10, 11, 12, 14, 57
predefined • 10, 12, 22
predefined function • 22
PREPARE • 64
PRIMARY • 17, 55
primary key • 17
primary key constraint • 17
PRIOR • 55
privilege • 9, 18, 54, 60, 61
PRIVILEGES • 58
procedure • 7, 8, 17, 19, 20, 21, 22, 23, 28, 36, 46, 49, 61
production rule • 31, 32, 34
property • 5, 12, 18
PUBLIC • 18

— Q —

<quantified comparison predicate> • 53
<quantified predicate> • 54
query • 10, 17, 34, 37, 53, 54, 56, 58, 59, 60, 62, 66, 67
<query expression> • 34, 37, 56, 58, 60
<query specification> • 53

— R —

READ • 21, 56
read-only • 21
READ ONLY • 56
read-write • 21
READ WRITE • 56
REAL • 52
REF • 12
referenced columns • 17
referenced table • 17
REFERENCES • 18, 54
reference type • 12, 62, 66
referencing columns • 17
referencing table • 17
referential constraint • 17, 60
REFERENTIAL_CONSTRAINTS • 59
RELATIVE • 55
REPEATABLE • 21
repertoire • 11, 15
representation • 5, 6, 9
requires • 5, 17, 37
reserved • 10, 43, 45
RESTRICT • 56, 57, 60, 65
RESULT • 22
result parameter • 22
returned value • 13
<return statement> • 61
REVOKE • 57

<right paren> • *40*, *41*
role • 18
role authorization • 18
<rollback statement> • 56
ROLLUP • 62, 66
<routine invocation> • 34, 36, 61
row • 5, 6, 9, 10, 12, 14, 16, 17, 23, 36, 37, 40, 51, 54, 55, 58, 61, 62, 63, 67
row type • 5, 12, 14, 16, 62, 67

— S —

schema • 5, 7, 8, 9, 10, 12, 14, 15, 18, 21, 22, 23, 26, 27, 28, 32, 33, 34, 35, 46, 47, 52, 56, 58, 59, 60
SCHEMA • 9, 56
<schema definition> • 52
schema routine • 18
scope • 1, 5, 6, 32, 36, 38, 65
search condition • 17, 18, 35, 37, 53, 54, 61
<search condition> • 37, 54
<searched case> • 58
SECOND • 12
SELECT • 18, 53, 54, 58, 61
<select list> • 53, 55
<select statement: single row> • 55
<select sublist> • 53
sequence • 5, 6, 11, 12, 15, 21, 23, 32, 35
serializable • 21
SERIALIZABLE • 21
SET • 56, 61
set function • 58, 59
set of subject routines • 34
shall • 6, 17, 32, 33, 34, 36, 38, 41, 42, 45, 46, 47
signature • 22
significant • 25
SIMILAR • 60
<simple case> • 58
simply contain • 34
simply contained in • 34
simply containing • 34
simply contains • 34
site • 5, 6, 7, 10, 12, 13, 14, 15, 16, 25, 36
SMALLINT • 52
<sort key> • 55
source • 14, 26
specific name • 14
specified by • 5, 8, 10, 12, 13, 16, 19, 22, 34, 35, 37, 43
specifies • 10, 12, 17, 19, 20, 21, 25, 26, 28, 29, 31, 32, 40, 41, 46, 49, 51, 63, 65
specify • 1, 8, 10, 11, 21, 25, 26, 27, 32, 33, 34, 35, 36, 37, 38, 51, 58, 61, 63
SQL-agent • 7, 8, 13, 19, 20, 21, 25, 27, 28
SQL-client • 7, 8, 10, 19, 20, 21, 23, 26, 27, 45, 46
SQL-client module • 7, 8, 10, 19, 20, 21, 23, 45, 46
<SQL conformance> • *40*, **41**
SQL-connection • 8, 23, 26, 27
SQL-data • 6, 7, 8, 9, 10, 13, 14, 16, 21, 23, 26, 33, 35, 46
<SQL edition> • ***40***, 41, 42
SQL-environment • 6, 7, 9, 21, 22, 25, 27, 37
SQL-implementation • 1, 5, 6, 7, 8, 9, 10, 13, 15, 19, 20, 21, 22, 25, 26, 27, 28, 40, 43, 45, 46, 65
SQL-invoked function • 22, 35
SQL-invoked procedure • 22
SQL-invoked routine • 14, 18, 21, 22, 34, 36, 61
<SQL-invoked routine> • 34, 36
<SQL object identifier> • **40**
SQL-path • 67
<SQL procedure statement> • 36
<SQL provenance> • ***40***
SQL routine • 22, 24, 27, 63, 67
SQL-schema • 5, 7, 9, 14, 21, 23, 26
SQL-server • 7, 8, 9, 14, 18, 21, 22, 23, 26, 27, 66
SQL-server module • 14, 18, 21, 22, 27, 66
<SQL-server module definition> • 66
SQL-session • 5, 8, 9, 10, 13, 18, 22, 23, 26, 27, 28, 58
SQL-session module • 22
SQLSTATE • 20, 56
SQL-statement • 6, 7, 8, 13, 19, 20, 21, 22, 23, 25, 26, 27, 28, 29, 33, 36, 45, 46
SQL-transaction • 5, 10, 13, 18, 21, 23, 26
<SQL variant> • ***40***
START • 61
state • 1, 6, 7, 8, 9, 10, 13, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 32, 33, 36, 37, 39, 43, 45, 46, 47, 52, 53, 54, 55, 56, 57, 58, 61, 62, 64
STATE • 20, 56, 65
status parameter • 20, 21, 33
subject routine • 34
subject table • 61
<subquery> • 56
substitutability • 16
SUBSTRING • 10, 52, 60
subtable • 16, 62, 67
subtype • 6, 16, 63, 67
<subtype treatment> • 63, 67
successful completion • 21
SUM • 54
supertable • 16
supertables • 16
supertype • 16
supertypes • 16
supplied • 47
syntax error or access rule violation • 34, 35

— T —

table • 5, 6, 7, 9, 10, 12, 13, 14, 15, 16, 17, 18, 22, 23, 26, 27, 29, 33, 34, 36, 37, 38, 39, 40, 51, 52, 53, 54, 56, 57, 58, 59, 60, 61, 62, 63, 66, 67
TABLE • 53, 56, 57
table constraint • 15, 17, 18, 37
<table definition> • 52
table descriptor • 37
<table name> • 34, 53
TABLES • 56
TABLE_CONSTRAINTS • 58
TABLE_PRIVILEGES • 58

target • 10, 13, 14, 33
temporary table • 10
TIME • 12, 57
TIMESTAMP • 12, 57
transaction • 5, 10, 13, 18, 19, 21, 23, 25, 26, 29, 61
TRANSACTION • 56, 61
transaction-initiating • 21
transaction manager • 25
transaction state • 21, 23, 26
translation • 15, 60
trigger • 17, 36, 61, 65
TRIGGER • 18, 61
trigger action time • 17
<trigger definition> • 36
triggered actions • 17
<triggered SQL statement> • 36
trigger event • 17
TRIM • 52
Type • 63, 67

— U —

UDT • 37, 62, 66
UNCOMMITTED • 21
UNION • 54, 57
unique columns • 17
unique constraint • 17
updatable • 10
UPDATE • 17, 18, 54, 56, 61, 62
<update statement: positioned> • 55
<update statement: searched> • 55
USAGE • 18, 58
USAGE_PRIVILEGES • 58
user-defined • 10, 14, 15, 16, 62, 63, 66, 67
user-defined type • 14, 63, 66, 67

— V —

valid • 3, 12, 37, 43
Value • 55
VALUES • 54, 58
variable • 10, 11, 13, 27
variant • 40
VARYING • 52, 57, 59
view • 9, 10, 14, 17, 25, 33, 34, 52, 53, 56, 57, 58, 59, 66
VIEW • 56, 57
view definition • 17, 33, 34, 53
<view definition> • 34, 53
viewed table • 10, 33
VIEWS • 56

— W —

WHERE • 58
WITH • 12, 18, 53, 54, 60, 61
with an intervening • 34
WITH CHECK OPTION • 53
with grant option • 18
WITH GRANT OPTION • 54
without an intervening • 34
WORK • 56
WRITE • 56

— X —

<199*x*> • *40*, **41**, 42

— Y —

YEAR • 12
year-month intervals • 12
YES • 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64

— Z —

ZONE • 12

Note 1, Jim Melton, 16-10-97 14:54:15
SC32 N00001