



**International
Standard**

ISO/IEC 19075-10

**Information technology —
Guidance for the use of database
language SQL —**

**Part 10:
SQL model (Guide/Model)**

**First edition
2024-10**

Reference number
ISO/IEC 19075-10:2024(en)

Full document details: ISO IEC 19075-10-2024

 **Documents**
© ISO/IEC 2024 Document Sharing Hub



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2024

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

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

Published in Switzerland

Contents	Page
Foreword.....	ix
Introduction.....	xi
1 Scope.....	1
2 Normative references.....	2
3 Terms and definitions.....	3
4 SQL model: origin and components.....	4
4.1 Relational model.....	4
4.2 The SQL model.....	5
4.3 The name “SQL”.....	7
4.4 Syntax conventions in SQL.....	7
5 Data integrity.....	9
5.1 Introduction to integrity.....	9
5.2 Transactions.....	9
5.3 Constraints.....	9
5.4 Predicates.....	10
5.5 Functional dependencies.....	11
6 Queries, expressions, and statements.....	12
6.1 Sets and multisets.....	12
6.2 Queries.....	12
6.3 Query expressions and query specifications.....	12
6.4 Table “shapes”.....	18
6.5 Subqueries.....	18
6.6 Views.....	18
6.7 SQL-statements.....	19
7 Data structure and metadata.....	20
7.1 Metadata.....	20
7.2 Metadata and SQL.....	20
7.3 SQL descriptors.....	20
7.4 Definition schema.....	20
7.5 Information schema.....	21
8 The SQL-environment and its components.....	22
8.1 Nomenclature of the components of the SQL model.....	22
8.2 Components of the SQL model.....	22
8.3 SQL-environment.....	22
8.4 SQL-agents.....	23
8.5 SQL-implementations.....	23
8.6 SQL-clients.....	24

8.7	SQL-servers.	24
8.8	SQL-connections.	25
8.9	SQL-sessions.	25
8.10	SQL-client modules, externally-invoked procedures, and embedded SQL.	25
8.11	Declarative and procedural statements.	27
8.12	Client-server model.	28
8.12.1	Introduction to the client-server model.	28
8.12.2	SQL-server modules.	28
8.13	SQL-statements.	29
8.14	Cursors.	30
9	Accessing SQL-data.	31
9.1	Introduction to SQL-data access.	31
9.2	“Interactive” SQL.	31
9.3	Module language.	31
9.4	Embedded SQL.	34
9.5	Dynamic SQL.	35
9.6	Other mechanisms to access SQL-data: Call-Level Interface (CLI).	36
9.7	Foreign servers and foreign-data wrappers.	36
10	Active databases and SQL.	37
10.1	Introduction to active databases.	37
10.2	Referential integrity and referential actions.	37
10.3	Triggers.	38
11	Transaction model.	39
12	Security model.	43
12.1	User identification.	43
12.2	Ownership of SQL-data.	43
12.3	Privileges and privileged objects.	43
12.4	Special considerations for views.	44
12.5	Definer’s and invoker’s rights.	44
13	Diagnostics model.	45
13.1	SQLSTATE.	45
13.2	Diagnostics area.	45
14	Data types.	47
14.1	Built-in data types.	47
14.1.1	Atomic data types.	47
14.1.2	Constructed data types.	48
14.2	User-defined types.	48
15	Schema creation and manipulation.	50
16	Other data models.	52
17	Conformance to ISO/IEC 9075.	53
Annex A	(informative) History of the SQL standard.	54
A.1	Background.	54
A.2	Further development.	56
Bibliography		57

Index.....59

Tables

Table	Page
1 Dirty read.....	39
2 Non-repeatable read.....	40
3 Phantom read.....	40

Figures

Figure	Page
1	Illustration of an SQL table. 6
2	Primary key — Foreign key relationships. 10
3	Cartesian product. 13
4	Filtering. 15
5	Grouping. 16
6	Projection. 17
7	First approximation of SQL-environment. 24
8	Second approximation of SQL-environment. 25
9	Relationship between SQL-client, SQL-client module, and SQL-statement. 26
10	Transformation from embedded SQL to externally-invoked procedure. 27
11	Minimizing client-server context changes. 29

Examples

Example	Page
1 Search condition.	11
2 Query specification.	12
3 Join based on equality.	14
4 Subquery use.	18
5 Example of direct invocation.	31
6 Module language example: SQL code fragment.	32
7 Module language example: COBOL code fragment.	32
8 Embedded SQL example.	34

Foreword

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

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

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

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

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

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

This document is intended to be used in conjunction with the following editions of the parts of the ISO/IEC 9075 series:

- ISO/IEC 9075-1, sixth edition or later;
- ISO/IEC 9075-2, sixth edition or later;
- ISO/IEC 9075-3, sixth edition or later;
- ISO/IEC 9075-4, seventh edition or later;
- ISO/IEC 9075-9, fifth edition or later;
- ISO/IEC 9075-10, fifth edition or later;
- ISO/IEC 9075-11, fifth edition or later;
- ISO/IEC 9075-13, fifth edition or later;
- ISO/IEC 9075-14, sixth edition or later;
- ISO/IEC 9075-15, second edition or later;
- ISO/IEC 9075-16, first edition or later.

ISO/IEC 19075-10:2024(en)

A list of all parts in the ISO/IEC 19075 series can be found on the ISO and IEC websites.

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

Introduction

This document describes the model of database language SQL, as defined primarily in [ISO/IEC 9075-1](#) and [ISO/IEC 9075-2](#). It supplies some background in the form of a brief history of the relational data model, plus a brief overview of some SQL features relevant to the SQL model.

The SQL model, of course, existed from the initial definition of the language, but was not explicitly recognized until the language became sufficiently complex that appropriate vocabulary and definitions were required to accurately specify all capabilities required.

This document summarizes the result of hundreds of hours of discussion and analysis that were required in order to create a coherent and useful definition of the SQL model of operation. This document briefly outlines the Relational model popularized by E.F. Codd. However, the fundamental structure and content of this document addresses SQL language facilities such as tables, column, relationships, and operations, not the Relational model itself.

Since SQL's adoption in 1986 as a database language standard, many extensions have been proposed, defined, reviewed, modified, and adopted by members of SQL-related standards committees across the world. While the SQL model herein addresses only a small subset of the SQL language's data management facilities, SQL's overall scope includes non-relational data types, object classes, multi-level data structures, and even DBMS-generated pointers to rows that appear as if they were user-based values. Supporting all all these SQL capabilities, there exists extensive and formally-defined language constructs that (when correctly implemented by SQL database implementers) provide a data definition and processing environment that is reliable and repeatable across all SQL standard-adherent implementations.

NOTE 1 — In this document, the terms “SQL” and “SQL language” are used as synonyms. The terms “SQL standard”, “ISO/IEC 9075”, and “the ISO/IEC 9075 series” are used as synonyms and refer to the standard for SQL [language], which is the ISO/IEC 9075 series of standards.

Information technology — Guidance for the use of database language SQL —

Part 10:

SQL model (Guide/Model)

1 Scope

This document describes the model of database language SQL as defined in [ISO/IEC 9075-1](#), [ISO/IEC 9075-2](#), and [ISO/IEC 9075-11](#). The meanings of and the relationships between various concepts of that model are described in text and illustrated graphically. Background in the form of some historical review and a brief overview of key SQL features is included.

NOTE 2 — In spite of the fact that the names of the ISO/IEC 9075 series of standards contain the phrase “database language”, the standards do not use the word “database” to describe the *thing* that SQL creates and on which it operates. The word “database” is used in many different contexts and has meanings wholly unrelated to the intent of the ISO/IEC 9075 series. Consequently, a variety of other terms are defined and used by the ISO/IEC 9075 series. The word “database” is frequently used in this document informally to mean “a collection of data managed by an SQL-implementation at any given time.”

2 Normative references

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

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

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

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

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 9075-1, ISO/IEC 9075-2, and ISO/IEC 9075-11 apply.

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

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <https://www.electropedia.org/>

4 SQL model: origin and components

4.1 Relational model

In 1970, Dr. E.F. Codd published [Relational Model \[32\]](#). Dr. Codd's premise was that "users of large data banks must be protected from having to know how the data is organized in the machine". The computer industry had years of experience with database management systems such as:

- IBM's Information Management System (IMS)[™] that used a hierarchical model to organize data;
- The Conference/Committee on Data Systems Languages and its CODASYL data model that managed relationships between data records using chains of "pointers" (often called a "network database system");
- The American National Standard [ANSI X3.133:1986 \[2\]](#) and its International Standard companion [ISO 8907:1987 \[1\]](#), which was a standardized specification of a CODASYL-like language;
- Cincom System, Inc.'s TOTAL[™], which was a database management system that combined hierarchical and network data management techniques.

Such systems offered high-performance access to data, particularly data that was closely related to data currently accessed, provided there were predefined, relevant links (or pointers) to that related data. Unfortunately, those links were *built into* the data as stored in the database system and it was not possible to easily reorganize them.

Codd's belief was that it was not necessary for application programs and other users to have intimate knowledge of the manner in which data was organized in persistent storage, nor to suffer the performance problems caused by a change in application requirements that rendered existing data relationships unhelpful.

The *relational model of data*, as Codd's work is universally known, separated the logical organization of data from its physical organization and used straightforward mathematical definitions to identify relationships among data on demand by application requirements.

The relational model defines collections of closely-related data as "relations" (sets of tuples); each "tuple" (set of elements) is therefore unique within its containing relation. Every relation is described by a metadata structure called a "relation scheme", which is a set of "attributes". Each attribute is identified by a unique name or by its ordinal position within the scheme, and has a datatype. In the relational model, each element of a tuple is mapped to a scheme attribute, and every tuple has the same number of elements as the scheme has attributes. Each element is therefore an "attribute value" that is defined by the data type of its corresponding attribute.

The mathematical underpinnings of the relational model are based on the definition of a handful of operations. There are two broad categories of these operations, commonly called "relational algebra" and "relational calculus".

The relational algebra specifies a number of simple operators, each of which transforms one or more source relations to a result relation. The operators include:

- *selection*, which transforms a source relation into a result relation by eliminating tuples that do not satisfy a specified predicate;
- *projection*, which transforms the source relation into a result relation by eliminating one or more attributes (more precisely, the result relation incorporates only the attributes named by the projection operator);

- *join* (also known as “Cartesian product”), which combines two relations into a result relation by combining every tuple in one relation with every tuple in the other relation.

There are numerous variations of join operation. Some of those variations are discussed briefly in [Subclause 6.3, “Query expressions and query specifications”](#).

Most resources, including Codd’s original paper, add a few more basic operators to those already listed. They are:

- *rename*, which transforms the source relation into a result relation by changing the names of one or more attributes to a specified name;
- *set union*, which combines two relations into a result relation by copying all tuples that are found in either or both relations;
- *set difference*, which combines two relations into a result relation by copying all tuples of one relation *except* those tuples that are found in the other relation.

Both set union and set difference require that the two relations be “union compatible”, meaning that they must have the same number of attributes and the names of the respective attributes must be the same. (Unstated, but implicit, the respective attributes must also be of the same, or a compatible, data type.)

Relational algebra has a kind of procedural aspect to it. Most application operations on a relational system that use a relational algebraic approach will first combine relations (joins, unions, differences) to create a new relation, then filter that new relation to eliminate tuples that fail to satisfy one or more predicates to create yet another new relation, then project that relation onto the desired set of attributes and create a final, result relation.

Relational calculus, by contrast, provides the ability for applications to express the desired result in a non-procedural manner, allowing the underlying software to determine the most effective (and, presumably, efficient) mechanisms to achieve that result.

Codd proved that relational algebra and relational calculus are logically equivalent and one can be transformed into the other. The primary justification for the ostensibly less-precise structure of relational calculus is to promote automatic *optimization* of database operations. That is, a database system, when presented with a relational calculus expression, can transform it into many possible equivalent relational algebra expressions based on minimizing data access times, numbers of operations, and so forth.

4.2 The SQL model

SQL is based on the relational model. However, SQL uses different terminology for its basic data structures, for at least two reasons: first, those basic data structures have minor but important differences from their relational model analogs; second, it was felt that the users of the language would prefer a less “technical-sounding” nomenclature than the mathematical terminology used in the relational model.

SQL’s analog of the relational model’s “relation” is called a *table*. A table is made up of *columns* and *rows*, corresponding respectively to the relational model’s attributes and tuples. [Figure 1, “Illustration of an SQL table”](#), illustrates an SQL table.

ISO/IEC 19075-10:2024(en)
4.2 The SQL model

EMPLOYEES					
EMP_ID	EMP_NAME	DEPT_ID	CATEGORY	AMOUNT	HIRE_DATE
E00147	John Philips	ENG	S	185300.00	2012-08-25
S00023	Stefan Schultz	MFG	H	23.00	2020-04-13
M00052	Marie Lacroix	MGT	S	224000.00	2008-11-10
F01598	Bill Hanson	ENG	S	106500.00	2014-04-30
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
J00215	Willem Peshma	MNT	H	16.25	2021-02-15
D01284	Mike Johnston	DES	S	96000.00	2018-06-17
S00007	Juan Muñoz	ENG	S	88750.00	2020-01-28
E02853	Kohji Fujita	FAC	H	14.80	2019-12-02

Figure 1 — Illustration of an SQL table

The most important — indeed, fundamental — differences between the relational model of data and the SQL model are:

- 1) In the relational model, every tuple in a relation is unique; that is, no two tuples are permitted to have the same value in every respective attribute. Relations are mathematical *sets*. By contrast, SQL permits *duplicate rows*, in which any number of rows are permitted to have the same values in respective columns.
- 2) The relational model requires that every attribute of a tuple have a (single) value of a specified data type; if the value of an attribute for a given tuple is not yet known, unavailable, or irrelevant (consider how the “year of marriage” of an unmarried person’s is recorded on a paper form), then some indication, such as a “default” value, must be provided. However, there are situations in which no such default value is sensible. Consequently, SQL defines a special kind of “value”, called a *null value*, that can be assigned to columns of any data type in any row of a table¹ to represent information that is unknown for any reason.
- 3) Comparisons of data in the relational model always results in a known result, true or false; that is, values are either equal or not equal, greater than or less than, one another. This is a direct result of the fact that every data object has to have a known value of a specified data type. SQL accommodates its null values by defining *three-valued logic*, in which the results of a comparison can be either true or false — or unknown. In SQL, if one compares a (non-null) value with a null value, the result is always unknown. Three-valued logic sometimes leads to results that some people find counter-intuitive, but it is mathematically provable to be correct; it also offers considerable computational power for applications that use it properly.

In the relational model, there may be no sequence (or order) to the attributes of a relation, and the relation, as a set of tuples, is by definition unordered. By contrast, in the SQL model there are circumstances in which the sequence of columns in a table is important, and there are operations that will form an application-specified sequence of the rows of a table.

SQL is a language that combines aspects of both relational algebra and relational calculus.

¹ In SQL, columns can be defined to prohibit null values, if desired.

As described in [Clause 6, “Queries, expressions, and statements”](#), SQL defines *query expressions* that formulate requests to the database system to identify information of interest to applications. Just as operations in the relational model accept one or more relations as sources and transform those operations to a new, result relation, SQL’s operations accept one or more tables as sources and transform them to new, result tables. Query expressions contain syntactic elements that correspond to the relational algebra operations select, project, join, union, difference, and others; they also contain elements that more closely resemble relational calculus.

In SQL, *persistent base tables* are tables that are “stored” in persistent storage. SQL also provides *virtual tables* that are “computed” upon demand by the evaluation of query expressions. Persistent base tables and virtual tables are both usable as sources to relational operations, usually without significant distinction; the most important distinction between the two are that persistent base tables are known by persistent names (table names), while virtual tables are often computed, used, and discarded without ever having been given a name.

If a particular virtual table’s definition is used frequently — that is, there is frequent need to compute the virtual table using an identical query expression — the definition is made persistent so that the virtual table can be referenced by name in the same manner as a persistent base table. A virtual tables with a persistent definition is called a *view*.

The SQL standard defines both the syntax and the semantics of the SQL language. The syntax is specified in a particular dialect of “extended BNF”² selected by the technical committees that defined the standard. The dialect of BNF used is described in [ISO/IEC 9075-1](#).

In the SQL standard’s specification of the semantics of the language, “general” rules describe an algorithm (possibly not the most efficient algorithm) that, if implemented, would produce the desired results. SQL-implementations are not required to implement that algorithm, as more efficient implementations are usually possible. The standard uses the word “effectively” to indicate that the intent of the algorithm is not to specify the actual implementation, but only the result that an implementation must achieve.

NOTE 3 — A brief history of the SQL language is provided in [Annex A, “History of the SQL standard”](#).

4.3 The name “SQL”

The name “SQL”, in the context of the ISO/IEC 9075 series, is not an acronym, but is properly pronounced “ess cue ell”. Some products that implement the standard pronounce that sequence of letters in their products’s names “sequel”; others use the phrase “structured query language” to describe their products. The original developers of the language that eventually became the SQL standard named their language “Structured English Query Language” and planned to use the shorter name “SEQUEL”, but discovered that term had already been trademarked.

4.4 Syntax conventions in SQL

The syntax of SQL, like that of most programming languages, includes “keywords”, special characters (“punctuation”), and identifiers (which are the names of data objects used in the language). Some keywords are reserved, meaning that they cannot be used as the names of data objects; other keywords are not reserved and are sometimes referred to as “noise” words (meaning that their use is not necessarily required for proper analysis of the intent, but whose presence can improve the readability for users).

In SQL, all keywords (reserved or not) may be “spelled” using any combination of upper-case and lower-case characters. For example, the reserved keyword “SELECT” may be written as “Select”, “select”, or even “sELeCT” without affecting its meaning or the ability of the SQL implementation to determine its usage. By convention in this document, keywords are usually spelled using all upper-case characters.

An identifier in SQL is either a “regular identifier” or a “delimited identifier”.

2 “BNF” is an initialism for “Backus Naur Form”, named for its inventors, John Backus and Peter Naur.

An ordinary identifier is required to start with a character that belongs to one of the Unicode character categories “Lu”, “Ll”, “Lt”, “Lm”, “Lo”, or “Nl”. Those categories identify characters that are, respectively, upper-case letters, lower-case letters, title-case letters, modifier letters, other letters, and “letter numbers”. The remainder of an ordinary identifier is permitted to be one of the categories allowed for the initial character, as well as characters that belong to one of the additional Unicode character categories “Mn”, “Mc”, “Nd”, and “Pc”, which respectively identify characters that are non-spacing marks, spacing combining marks, decimal numbers, and connector punctuations. In the SQL standard, lower-case alphabetic characters contained in ordinary identifiers are implicitly transformed to their corresponding upper-case characters.

For example, the ordinary identifier “street_name” may be written as “Street_Name”, “Street_name”, or “street_Name”; all variations correctly identify the same SQL object, the canonical spelling of which is “STREET_NAME”. All such spellings are interpreted in the SQL standard as representing the same SQL object. In this document, identifiers are often spelled using only lower-case characters, but some text uses mixed-case or only upper-case characters; the choice is usually determined by what concept the text discusses and what it is desired to emphasize.

NOTE 4 — The transformation of lower-case alphabetic characters to their corresponding upper-case characters is not necessarily a one-to-one transformation. For example, an identifier written by a German programmer that is spelled “straßen_name” has a canonical spelling “STRASSEN_NAME”.

By contrast, a delimited identifier is a syntax element that begins and ends with a “double quote” character “”” (equivalent to Unicode character U+0022 and contains a sequence of characters (other than a double quote character); no character in that sequence is permitted to belong to any of the Unicode character categories “Cc”, “Cf”, “Cn”, “Cs”, “Zl”, or “Zp”. (Those categories are, respectively, control characters, formatting characters, unassigned characters, surrogates, line separators, and paragraph separators.) If the name of an SQL object is required to incorporate a double quote character, that is represented as “”” (two consecutive double quote characters).

NOTE 5 — In the SQL standard, identifiers must not exceed a maximum length (number of characters) that is determined by the SQL-implementation. SQL-implementations must support identifiers of at least a minimum number defined by the standard. That minimum number, in editions of the SQL standard from 1986 through 2023, is 18 characters. In future editions of the SQL standard, it is possible that the minimum length of identifiers that implementations are required to support will be changed to some other value, such as 128 characters.

The names of SQL objects (identifiers) have additional restrictions placed on them. One restriction is that certain names are “qualified” by the names of other objects. For example, the names of the columns of an SQL table are implicitly qualified by the name of the table: the salary column of the employees table is, in its qualified form, employees.salary. Table names are further qualified by the name of the “container” in which tables are defined, called the “schema”. If the employees table is contained in a schema named headquarters, then the name of the table is effectively headquarters.employees and the name of the salary column is effectively headquarters.employees.salary.

NOTE 6 — In the SQL standard, schemata are specified to exist in the context of another kind of SQL object, called a “catalog”. Few SQL-implementations explicitly support catalogs; in practice, that means that the catalog name cannot be specified as a qualifier for schema names (in effect, only a single, unnamed, catalog is implied).

5 Data integrity

5.1 Introduction to integrity

In any database, including an SQL database, it is highly likely that much of the data relates closely to other of the data. The value of that data is highly dependent on it being correct, or at least as correct as possible. When two or more pieces of data are related to one another, it is imperative that their relationships and correctness are correlated.

SQL provides a number of facilities to help ensure that data is correct, that it remains correct, and that relationships between data remain correct. Among these capabilities are the implementation of transactions and the provision of integrity constraints on data and their relationships.

5.2 Transactions

A transaction is defined in computer science to be an atomic unit of work that either succeeds entirely or fails entirely. If any operation performed as part of a transaction fails, then all previous operations performed as part of the same transaction are reversed, the failure is reported to the application, and the data is restored to the values it had before the transaction was initiated.

The semantics of the SQL standard require that all changes made to data managed by an SQL-implementation are done as part of a transaction. It also assumes that every transaction completes before another transaction is initiated. When transactions behave in this manner, they are referred to as “serialized” transactions, meaning that they are initiated and terminated in a sequential fashion, without overlap.

Database systems normally do not actually serialize transactions, but provide *serializable* transactions — transactions that behave as though they were executed sequentially, but that overlap in time. In order for transactions that overlap to behave as though they were executed sequentially, their operations must be *isolated* from those of other, concurrent transactions. There are many implementation mechanisms that can be used to provide transaction isolation, but those are beyond the scope of this document. The SQL standard permits SQL-implementations to provide varying levels of transaction isolation, ranging from fully serializable to a point where the data “seen” by one transaction can be affected by changes caused by another transaction *even if that other transaction eventually fails*. Transactions and isolation levels are discussed in greater detail in [Clause 11, “Transaction model”](#).

In order to ensure the highest level of data accuracy and consistency, application programs start transactions as serializable transactions. Applications that have lower data accuracy and consistency requirements are able to start transactions with lower levels of isolation.

5.3 Constraints

SQL provides capabilities that allow implementations to continuously validate that the data they manage satisfy defined value restrictions, including specific values or ranges of values and relationships between multiple values. Those capabilities are generically known as *integrity constraints*.

SQL defines several kinds of integrity constraint: unique constraints, including primary key constraints; referential constraints; and table check constraints. Most table check constraints are defined in the context of a single table, but SQL provides a variation, called an *assertion*, that is defined with a larger scope than a single table.

Unique constraints are used in the definition of a table to specify that the value of a specified column or group of columns of each row of the table is required to be unique among all rows of the table. Ordinary unique constraints do not prohibit values in the specified column or group of columns from being the

null value; all such null values are ignored for the purposes of determining unique values. SQL permits multiple unique constraints to be defined for tables, but does not require that every table have a unique constraint.

A *candidate key* is a unique constraint that identifies a column or group of columns that is defined to prohibit null values. Special syntax (PRIMARY KEY) is used to define a particular kind of candidate key that both requires the value of the designated column, or the combination of values of the designated group of columns, must be unique among all rows of the table and are also prohibited from having the null value. No table is permitted to have more than one primary key, but tables may have no, one, or multiple candidate keys.

Frequently, data in one table is required to be closely associated with data in a second table. That close association is often indicated by requiring that the value of one column or group of columns in every row of the first table be equal to the value of some column or group of columns in some row of the second table. If the second table — called the *referenced table* — has been defined to have a unique constraint comprising that column or group of columns, then the first table — called the *referencing table* — can be defined to have a *foreign key* that identifies the referenced table. See Figure 2, “Primary key — Foreign key relationships”, for an illustration of this relationship.

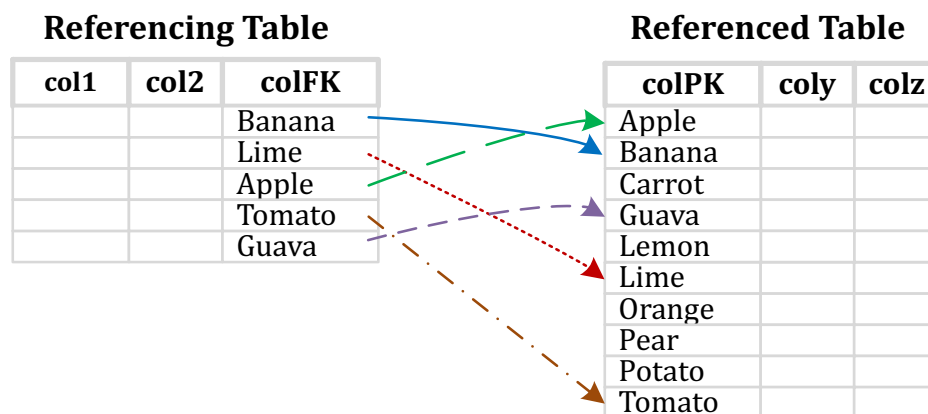


Figure 2 — Primary key — Foreign key relationships

The referenced table is sometimes informally called the “parent table” and the referencing table the “child table”.

5.4 Predicates

The SQL standard provides facilities that allow applications to compare values to other values and to test a variety of situations. Those facilities are known as “predicates”. A predicate is an expression that can take at most three values: *True*, *False*, and (under some conditions) *Unknown*.

In SQL, predicates are most often used in the `WHERE` clause of a query specification (see Subclause 6.3, “Query expressions and query specifications”), but are used in other places as well (such as the `FROM` clause and in constraints). The most common form of predicate is the *comparison predicate*, which allows comparison of values based on their equality or lack thereof, as well as based on one value being less than or greater than the other. Other predicates permit testing whether a value is the SQL null value or not, whether a virtual table has any rows or not, whether a specific value is unique among a collection of values, and numerous other tests.

ISO/IEC 19075-10:2024(en)
5.4 Predicates

Predicates are often combined into complex tests using logical operators such as AND, OR, and NOT. Such combinations are known as *search conditions* in the SQL standard. [Example 1, “Search condition”](#), provides an example of such a combination.

Example 1 — Search condition

```
employees.dept_id = departments.dept_id AND  
( employee.salary > 100000 OR  
  employee.salary IS NULL )
```

5.5 Functional dependencies

A *functional dependency* is a relationship between identifying columns of a table and sets of columns of that table. Such sets are concerned with “base table unique constraint sets” and “base table primary key sets”. In SQL terms, it is a set of *known not null* columns of a unique constraint (of a base table), or a *primary key constraint* (of a base table), respectively.

The logic of inferring known functional dependencies is beyond the scope of this document, but described in detail in [Subclause 4.26, “Functional dependencies”](#), of ISO/IEC 9075-2:2023 [14].