Technical Report

ISO/IEC TR 19583-21

Information technology — Concepts and usage of metadata —

Part 21: 11179-3, -31, -32 Data model in SQL

Technologies de l'information — Concepts et utilisation des métadonnées —

Partie 21: Modèle de données en SQL 11179-3, -31, -32

Second edition
2025-03

Reference number
ISO/IEC TR 19583-21:2025(en)

Contents

Page

Foreword

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

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

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC 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 www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

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

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

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

This second edition cancels and replaces the first edition (ISO/IEC 19583-21:2022), which has been technically revised.

The main changes are as follows:

— examples have been added to instatiate the metamodel of the latest version of the ISO/IEC 11179 series (Annexes B and C have been added).

A list of all parts in the ISO/IEC 19583 series can be found on the ISO and IEC websites.

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

Introduction

The ISO/IEC 11179 series[1] provides a specification for a registry in which information about metadata can be recorded and maintained.

The metamodel to instantiate such a registry is expressed in text as a conceptual model. This conceptual model is illustrated with a series of diagrams which use the class diagram notation from the Unified Modeling Language (UML).[2][3]

Implementers and users of the registries described in ISO/IEC 11179 require further guidance to turn the conceptual models into concrete instantiations. This document provides a possible instantiation of the registry metamodel specified in ISO/IEC 11179-3[4], ISO/IEC 11179-31[5] and ISO/IEC 11179-32[6] using the SQL database language as specified in the ISO/IEC 9075 series.[7]

This specimen instantiation is provided to increase the understanding of ISO/IEC 11179-3, ISO/IEC 11179-31 and ISO/IEC 11179-32, and hence, to promote its adoption.

This document is not intended to replace the UML version but rather serves as a complement, providing SQL statements that describe the ISO/IEC 11179 metamodel. It aims to facilitate the application of ISO/IEC 11179 in a database or related environment.

Information technology — Concepts and usage of metadata —

Part 21:
11179-3, -31, -32 Data model in SQL

1 Scope

This document provides a possible instantiation of the registry metamodel specified in ISO/IEC 11179-3, ISO/IEC 11179-31, ISO/IEC 11179-32 using the SQL database language as specified in ISO/IEC 9075-2.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

No terms and definitions are listed in this document.

ISO and IEC maintain terminology 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 The relationship among ISO/IEC 11179-3, ISO/IEC 11179-31 and ISO/IEC 11179-32

4.1 Overview

The relationships among these parts are mutually dependent, and they collectively form a complete metadata registry system. ISO/IEC 11179-3 provides infrastructure support for the entire registry. ISO/IEC 11179-31 and ISO/IEC 11179-32 extend the functionality of the metadata registry by registering data specifications and concept systems, further enriching the metadata information stored in the registry. These parts work together to enable the metadata registry to effectively support data management and the discovery, understanding, and utilization of data assets.

4.2 ISO/IEC 11179-3 — Metamodel for registry common facilities

ISO/IEC 11179-3 specifies the core metamodel for a Metadata Registry (MDR), but it does so in a generic way, so that the resulting registry could be used to register anything, not just metadata.

The registry metamodel is not a complete description of all the metadata an organization might wish to record. Therefore, the model is designed to be extended if required. However, extensions are, by their nature, not part of ISO/IEC 11179-3.

Some extensions to the ISO/IEC 11179-3 metamodel are specified in other parts of ISO/IEC 11179, such as:

— ISO/IEC 11179-31: Metamodel for data specification registration;

— ISO/IEC 11179-32: Metamodel for concept system registration;

— ISO/IEC 11179-33: Metamodel for data set registration;

— ISO/IEC 11179-34: Metamodel for computable object registration;

— ISO/IEC 11179-35: Metamodel for model registration.

4.3 ISO/IEC 11179-30 — Basic attributes of metadata

ISO/IEC 11179-30[8] describes basic attributes for data elements and related metadata for use in contexts where a full metadata registry is not appropriate, such as standards documents. It is limited to a set of basic attributes for: data elements, data element concepts, value domains, conceptual domains and other related classes.

4.4 ISO/IEC 11179-31 — Metamodel for data specification registration

ISO/IEC 11179-31 provides a specification for an extension to a Metadata Registry (MDR), as specified in ISO/IEC 11179-3, in which metadata that describe data elements and associated concepts, such as data element concepts, conceptual domains and value domains can be registered.

The specification in ISO/IEC 11179-31, together with the relevant clauses of the specification in ISO/IEC 11179-3, provides the ability to record metadata about:

— data elements, units of measure and derivation rules;

— data element concepts and associated object classes and properties;

— conceptual domains and value meanings;

— value domains, value domain subsets and permissible values.

ISO/IEC 11179-31 is applicable to the formulation of data representations, concepts, meanings and relationships to be shared among people and machines, independent of the organization that produces the data. It is not applicable to the physical representation of data as bits and bytes at the machine level.

4.5 ISO/IEC 11179-32 — Metamodel for concept system registration

ISO/IEC 11179-32 provides a specification for an extension to a Metadata Registry (MDR), as specified in ISO/IEC 11179-3, in which metadata that describe concept systems can be registered.

The specification in ISO/IEC 11179-32, together with the relevant clauses of the specification in ISO/IEC 11179-3, provides the ability to record the following metadata:

— concept systems and associated concepts;

— relations among concepts in a concept system;

— assertions about concepts in a concept system.

The metamodel in ISO/IEC 11179-32 is intended to support the full description of a concept system, including ontologies.

5 Overview of the relationship between UML Class Diagrams and SQL

The Unified Modeling Language (UML) provides a family of graphical notations that can be used in the analysis and design of software systems. The UML is under the control of the Object Management Group (OMG) and, as such, it is (a) a relatively 'open' standard, and (b) firmly rooted in the object-oriented paradigm for software engineering. The UML is now at Version 2 and is the subject of two international standards: ISO/IEC 19505-1 and ISO/IEC 19505-2.

Within the UML, the Class Diagram notation is used to represent information (and, hence, data) requirements for a particular 'universe of discourse', a business area or the scope of a proposed information system.

A UML Object is often defined as a:

construct within a system for which a set of attributes and operations can be specified.

Whilst this is a reasonable definition within the context of object-oriented system development, a more appropriate definition of an Object for the purposes of this document is a:

representation of something of interest within the universe of discourse about which information needs to be recorded.

An Object Class in both contexts can then defined as a:

definition of a set of Objects that share the same attributes, associations, and operations.

NOTE Although UML Class diagrams allow the specification of operations on the classes, the ISO/IEC 11179 series data models do not specify operations on their classes, so this document does not specify any operations either.

The Database Language SQL is a, largely, declarative language used to manage structured data held in a database under the control of a Relational Database Management System (RDBMS). As such, it was originally based on Edgar F. Codd's relational model of data published in 1970,[9] but its scope has grown over the years. SQL is the subject of the ISO/IEC 9075 series. Most commercial SQL products, however, deviate from the standards to some extent, some more than others.

6 Generating the SQL for the metamodel

6.1 Overview

The UML (and the Class Diagrams, in particular) and the SQL database language exist in two separate programming paradigms and there is, therefore, no direct translation from one (the UML) to the other (SQL). There are, however, approaches that can be taken to achieve a translation. This document uses one of those approaches to generate a set of SQL statements to instantiate the metadata registry metamodel, where the SQL statements enable easy reference back to the original UML Class Diagram and text of the metamodel. This is achieved by using names for the SQL objects that reflect the names of the UML artefacts and, also, by embedding comments referencing the metamodel within the SQL statements.

6.2 General principles for the translation of a UML Class diagram into SQL statements

It is good practice to distinguish between SQL keywords and the names given to the SQL objects. One convention is to use UPPER CASE for the keywords and lower_case (using snake case) for the object names.

Each UML class is represented by an SQL table. To make correlation to the model easier, the name of a table that represents a class is the same as that of the class.

Each composite datatype is also represented by an SQL table. To make correlation to the model easier, the name of the table is the same as that of the datatype.

Each single-valued attribute of a class or a composite datatype is represented by a column in the appropriate table. The name of this column is the same as that of the attribute in the class or datatype. The datatype of an attribute column is intuitively selected to be similar to that of the datatype of the attribute.

If the datatype of an attribute of a class is another class or a composite datatype, the column that represents that attribute is additionally declared as a foreign key column referencing the relevant table that represents the class or composite datatype.In this case, the column name is suffixed with "_id" to reflect the fact that it contains a reference to the data, not the data itself.

Where an attribute is multivalued (that is, it has a multiplicity of [0..*] or [1..*] in the UML diagram) there are two possible instantiations available. These are the following.

a) Use one of the collection types, MULTISET or ARRAY, available in SQL.

b) Create a new table, a characteristic table, to hold the multiple values, with each row in the table having a foreign key referencing the kernel (prime) table and one of the values.

In object-orientation, and, hence, UML, there is no equivalent of the SQL primary key, so each table that represents a class or a composite datatype has an additional column that is used as a surrogate identifier. This column then becomes the primary key for the table.

UNIQUE or CHECK constraints may be added to a table where required. The latter are used, for example, to control the valid values for a column or to control which columns should, or should not, take values in different circumstances.

Specialization hierarchies (superclasses and their subclasses) can be instantiated in one of two ways using SQL structures.

1) It is possible to integrate all the classes (the superclass and its subclasses) into one table, named after the superclass, which includes a column for each attribute of the superclass, with those columns representing mandatory attributes being declared NOT NULL, and columns for each of the attributes of the subclasses. None of these subclass columns are declared as NOT NULL, irrespective of whether they are mandatory or optional within their subclass. An additional column, often called a discriminator column, with possible values representing each of the subclasses and a CHECK constraint are provided to manage which columns are populated for each subclass. Where a subclass is related to another class via a one-to-many association, such that there is a foreign key in the table representing the other class, a cross-table constraint is needed to ensure that a row in the other table will only exist if the value of the discriminator column represents the relevant subclass in the inheritance relationship.

2) Another possibility is the creation of a separate table for the superclass and the creation of additional tables for each of the subclasses. When the hierarchy is complete, disjoint and static (the normal situation in most models), a fully mandatory 'one-to-one' relationship is provided between the table representing the superclass and each table representing the subclasses. To achieve this, a subclass type column is added, which has a fixed value in each subclass type, but the value of which in the superclass is set to match the corresponding subclass. Although this column does not form part of the primary key of either the superclass or subclass tables, it is appended to the primary key in the referential constraints that enforce the subclassing. This ensures that for each row in the superclass table there is a corresponding row in one of the subclass tables, and that there is no duplication of primary key values across the tables representing the subclasses.

There are a number of different approaches that can be used when translating UML Class Diagram associations into SQL. Since each association in the metadata registry metamodel is named, the approach used in this document is to create a table for each association, with the table named with the name of the association.

Some many-to-many associations are annotated with association classes. These association classes also become tables.

Since the ISO/IEC 11179 metamodel does not specify operations on the UML classes, this document does not specify how operations can be specified in SQL.

6.3 Specific approaches taken for the translation of the metadata registry metamodel

6.3.1 Overview

The following subclauses provide specific detail about the translation of the metamodel artefacts, where the information in 6.2 is either not applicable or insufficient.

6.3.2 Obligations

In the metamodel the obligations applicable to each attribute or association are described as one of "Mandatory", "Conditional" or "Optional", with these obligations being enforced if, and only if, the Registration Status of the associated metadata item is Recorded or higher, that is, if the Registration Status of the associated item is one of "Recorded", "Qualified", "Standard" or "Preferred Standard". The obligations are not enforced if the Registration Status of the associated item is one of "Candidate", "Incomplete", "Retired" and "Superseded".

Any registry instantiation has to be able to register items with a lower Registration Status than “Recorded”, and the obligations cannot, therefore, be simply enforced.

The example SQL instantiation allows the attributes and associations to be optional so that items with a Registration Status lower than “Recorded” can be accommodated. The obligations applicable to items with a Registration Status of “Recorded” or higher will, therefore, need to be enforced in the registry application as opposed to the register (the database) itself.

6.3.3 Translation of datatypes

The datatypes used in the metamodel can be considered to be ‘primitive’ or more complex.

The primitive datatypes are translated as described in Table 1.

Table 1 — Translation of the primitive datatypes

Metadata registry metamodel datatype	Examples or Comment	SQL Datatype
Boolean		This simply translates as a column of type BOOLEAN
Date		This simply translates as a column of type DATE
Datetime		This simply translates as a column of type TIMESTAMP
Integer		This simply translates as a column of type INTEGER
Notation	XCL Common Logic, OWL-DL XML	This translates as a column of type CHARACTER VARYING (2500)
Sign	This could be a bit string, but, at a minimum, String must be supported.	This translates as a column of type CHARACTER VARYING (2500). If the SQL implementation supports the BINARY LARGE OBJECT type, this could be used instead.
String		This translates as a column of type CHARACTER VARYING (255)
Text		This translates as a column of type CHARACTER VARYING (2500). If the SQL implementation supports the CHARACTER LARGE OBJECT type, this could be used instead.

The more complex datatypes are translated as described in Table 2.

Table 2 — Translation of the more complex datatypes

Metadata registry metamodel datatype	Examples or Comment	SQL Datatype
Natural_Range	0, 1, 2, 1..2, 2..8, 0..*, 3..*	This is instantiated with three columns, one INTEGER column for the lower bound, one INTEGER column for the fixed upper bound, and a CHARACTER column, defaulting to ‘many’, for the many upper bound. The columns are then managed with a CHECK constraint.
Value	This represents a value of any of the types listed above	This is instantiated using many different columns, one, or more, for each of the datatypes listed and then a CHECK constraint implemented to ensure that only one datatype is represented.
Phone_Number		A table is created (named cdt_phone_number) with columns as specified in ISO/IEC 19773[10]; the table has a surrogate primary key of datatype INTEGER. Tables representing classes that have attributes specified with this datatype have foreign keys that reference cdt_phone_number.
Postal_Address		A table is created (named cdt_postal_address) with columns as specified in ISO/IEC 19773[10]; the table has a surrogate primary key of datatype INTEGER. Tables representing classes that have attributes specified with this datatype have foreign keys that reference cdt_postal_address.

6.3.4 Translation of the basic classes

Each basic class in the metamodel is represented by a table, with the name prefixed by cls_. Each table representing a basic class has a surrogate primary key of type INTEGER.

6.3.5 Translation of the remaining classes

Each of the remaining classes in the metamodel is represented by a table, with the name prefixed by cls_. Each table representing a class has a surrogate primary key of type INTEGER.

6.3.6 Translation of specialization hierarchies

Each superclass and its subclasses are represented by separate tables. Those tables that represent subclasses inherit the surrogate primary key of the superclass, either as the primary key or as an additional column, with that inherited primary key also being declared as a foreign key. Keys and constraints are normally included when the tables are created.

6.3.7 Translation of the association classes

Each association class in the metamodel is represented by a table, with the name prefixed by asscls_. Each table representing an association class has a surrogate primary key of type INTEGER, a column, or columns representing the attributes of the association class, and two additional columns: one each for the roles of the association class, both of which have datatype INTEGER and are specified as foreign keys referencing the tables representing the relevant classes.

6.3.8 Translation of the attributes of the classes

Where the metamodel datatype of the attribute is listed in the first column of Table 1, the attribute is translated as a single column of the relevant table representing the parent class, with a datatype as specified in the third column of Table 1.

Where the metamodel datatype of the attribute is listed in the first column of Table 2, the attribute is instantiated as specified in the third column of Table 2.

Where the attribute is a multivalued attribute, there are three options available.

— Where the metamodel datatype is specified in Table 1 and the values are unordered, a separate characteristic table, prefixed mva_, is created with a column for the multivalued attribute specified with the SQL datatype identified in Table 1. The second column in Table 1 is a foreign key column referencing the table representing the class that specifies the attribute. Both columns are declared as the primary key. If the SQL implementation supports the MULTISET collection type, this can be used instead of creating the characteristic table.

— Where the datatype is specified in Table 1 and the values are specified as being ordered, or it makes sense to order them, a separate characteristic table, prefixed mva_, is created with a column for the multivalued attribute specified with the SQL datatype identified in Table 1. The second column in Table 1 is a column of datatype INTEGER named 'priority' to indicate the order of the value. The third column in this table is a foreign key column referencing the table representing the class that specifies the attribute. All three columns are declared as the primary key. If the SQL implementation supports the ARRAY collection type, this could be used instead of creating the characteristic table.

— Where the datatype is specified in Table 2, a separate characteristic table, prefixed mva_, is created with a column for the multivalued attribute specified with the datatype INTEGER and as a foreign key referencing the table representing the complex datatype. The second column in this table is a foreign key column referencing the table representing the class that specifies the attribute. Both columns are declared as the primary key.

Where the datatype of the attribute is specified as another class specified in the metamodel, the attribute is instantiated as a column specified with the datatype INTEGER and as a foreign key referencing the table representing the class that is specified as the datatype in the metamodel. Where feasible, this column is included in the CREATE TABLE statement. If this is not feasible, later ALTER TABLE statements are used to add the column and the foreign key.

6.3.9 Translation of the associations

Each association is one of three basic types, as follows:

— A zero or one-to-many association: In this case, a column is included in the table representing the class at the 'many' end of the association, with a referential constraint to the table representing the class at the other end of the association. The referential constraint is named with the name of the association prefixed by ass_.

— A many-to-many association: In this case, a separate association table is created, named with the name of the association prefixed by ass_, with two columns: one each for the roles of the association, both of which have datatype INTEGER and are specified as foreign keys referencing the tables representing the relevant classes. Both columns are declared as the primary key of the association table.

— A one-to-one association: In this case, a separate association table is created as for the many-to-many association, but, in addition, each column is declared with a UNIQUE constraint to enforce the one-to-oneness.

6.3.10 Cross-table constraints

Because obligations specified in the metamodel are only applicable if, and only if, the Registration Status of the associated metadata item is Recorded or higher, very few of the constraints needed to enforce the obligations are included in the example SQL.

No cross-table constraints are specified in this document.

7 Example SQL for instantiation of the metamodel

The complete set of SQL statements needed to provide this example SQL instantiation is contained in Annexes A, B and C.

The statements are provided in the following order.

a) CREATE TABLE statements to create the tables to represent the two complex datatypes, the Phone_Number class and the Postal_Address class.

b) CREATE TABLE statements to create the tables to represent the basic classes, with additional characteristic tables to represent multi-valued attributes where needed.

c) CREATE TABLE statements to create the tables to represent the remaining classes. These tables are created grouped by the metamodel regions of the ISO/IEC 11179 part from which they are sourced. Additional characteristic tables are created to represent multi-valued attributes where needed.

d) CREATE TABLE statements to create the tables to represent the association classes. These tables are specified following the class tables within each metamodel region grouping, and are specified in alphabetical order within each group. Additional characteristic tables are created to represent multi-valued attributes where needed.

e) CREATE TABLE statements to create the tables to represent the metamodel associations.

This set of SQL statements does not provide the most optimal instantiation of a database for a metadata registry, but it does provide an instantiation that can easily be traced back to the metamodel.

Annex A
(informative)

Example SQL to instantiate the ISO/IEC 11179-3 metamodel

A.1 Predefined types metamodel

```
/* -------------------------------------------------------------------------- */
/* create tables for predefined types                                         */
/* -------------------------------------------------------------------------- */
CREATE TABLE cdt_phone_number
  (
    /* surrogate primary key */
    phone_number_id                    INTEGER PRIMARY KEY                ,
    /* columns to represent attributes */
    international_numbering_plan_prefix CHARACTER VARYING(255)            ,
    country_code                       CHARACTER VARYING(255)             ,
    city_code                          CHARACTER VARYING(255)             ,
    local_number                       CHARACTER VARYING(255) NOT NULL    ,
    extension                          CHARACTER VARYING(255)
  );

CREATE TABLE cdt_postal_address
  (
    /* surrogate primary key */
    postal_address_id                  INTEGER PRIMARY KEY                ,
    /* columns to represent attributes */
    sub_building_name                  CHARACTER VARYING(255)             ,
    building_name                      CHARACTER VARYING(255)             ,
    throughfare                        CHARACTER VARYING(255)             ,
    dependent_locality                 CHARACTER VARYING(255)             ,
    post_town                          CHARACTER VARYING(255)             ,
    region                             CHARACTER VARYING(255)             ,
    postcode                           CHARACTER VARYING(255)             ,
    country                            CHARACTER VARYING(255)
  );
```


A.2 Basic classes metamodel

```
/* -------------------------------------------------------------------------- */
/* create tables for basic classes                                            */
/* -------------------------------------------------------------------------- */
CREATE TABLE cls_individual
  (
    /* surrogate primary key */
    individual_id                      INTEGER PRIMARY KEY                ,
    /* columns to represent attributes */
    name                               CHARACTER VARYING(255) NOT NULL    ,
    "title"                            CHARACTER VARYING(255)             ,
    mail_address_id                    INTEGER
      REFERENCES cdt_postal_address (postal_address_id)
    /* the multi-valued attributes email_address and phone_number */
    /* instantiated as the characteristic tables mva_individual_email_addresses */
    /* and mva_individual_phone_numbers */
  );

CREATE TABLE mva_individual_email_addresses
  (
    /* column to represent multi-valued attribute */
    email_address                      CHARACTER VARYING(255)             ,
    /* identification of the individual this email address belongs to */
    owning_individual_id               INTEGER
      REFERENCES cls_individual (individual_id)                           ,
```