— registered XML Schema

Append this paragraph USAGE privileges on registered XML Schemas are granted or revoked by implementation-defined means.

4.8 SQL-sessions

This Subclause modifies Subclause 4.37, "SQL-sessions", in ISO/IEC 9075-2.

4.8.1 SQL-session properties

This Subclause modifies Subclause 4.37.3, "SQL-session properties", in ISO/IEC 9075-2.

Insert after 6th paragraph An SQL-session has an XML option that is used to identify the <document or content> option needed in the implicit invocation of XMLSERIALIZE and XMLPARSE operators during the execution of <preparable statement>s that are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement>. The XML option is initially set to an implementation-defined value, but can subsequently be changed by the successful execution of a <set XML option statement>.

Augment the 13th paragraph

— The current XML option.

4.9 XML namespaces

This part of ISO/IEC 9075 references certain XML namespaces that are defined by the World-Wide Web Consortium or by this standard. Each XML namespace is referenced using an XML namespace prefix. The XML namespace prefixes and their definitions are shown in Table 2, "XML namespace prefixes and their URIs".

Table 2 — XML namespace prefixes and their URIs

XML namespace prefix	XML namespace URI
`xs`	`http://www.w3.org/2001/XMLSchema`
`xsi`	`http://www.w3.org/2001/XMLSchema-instance`
`sqlxml`	`http://standards.iso.org/iso/9075/2003/sqlxml`

A conforming implementation is not required to use the XML namespace prefixes **`xs`**, **`xsi`**, or **`sqlxml`** to reference these XML namespaces, but whatever XML namespace prefix it uses shall be associated with the proper URI.

The XML namespace identified by the XML namespace prefix "`sqlxml`" is normatively defined in Clause 22, "The SQL/XML XML Schema".

A resource containing the XML Schema definition of the XML namespace identified by the XML namespace prefix "`sqlxml`" (that is, a file containing an XML Schema document) has been made available on the World Wide Web. The URI of that resource is:

```
http://standards.iso.org/iso/9075/2003/sqlxml.xsd
```

It is intended that the contents of that file be identical to the contents of Clause 22, "The SQL/XML XML Schema". This file has been created for the convenience of the implementors of this part of ISO/IEC 9075.

4.10 Overview of mappings

This International Standard defines mappings from SQL to XML, and from XML to SQL. The mappings from SQL to XML include:

— Mapping SQL character sets to Unicode.

— Mapping SQL <identifier>s to XML Names.

— Mapping SQL data types (as used in SQL-schemas to define SQL-schema objects such as columns) to XML Schema data types.

— Mapping values of SQL data types to values of XML Schema data types.

— Mapping an SQL table to an XML document and an XML Schema document.

— Mapping an SQL schema to an XML document and an XML Schema document.

— Mapping an SQL catalog to an XML document and an XML Schema document.

The mappings from XML to SQL include:

— Mapping Unicode to SQL character sets.

— Mapping XML Names to SQL <identifier>s.

4.10.1 Mapping SQL character sets to Unicode

For each character set *SQLCS* in the SQL-environment, there shall be a mapping *CSM* of strings of *SQLCS* to strings of Unicode, as defined in [Unicode]. In this part of this International Standard, "Unicode" refers to the character repertoire named "UCS". The mapping *CSM* is called *homomorphic* if for each nonnegative integer *N*, there exists a nonnegative integer *M* such that all strings of length *N* in *SQLCS* are mapped to strings of length *M* in Unicode. *CSM* is implementation-defined. However, if any Unicode code point is mapped to a character that is not a valid XML character, an exception condition is raised.

NOTE 11 — The entity references **`&lt;`**, **`&amp;`**, **`&gt;`**, **`&apos;`**, and **`&quot;`**, as well as character references, as defined by [XML] are regarded as each representing a single character in XML, and do not pose an obstacle to defining homomorphic mappings.

4.10.2 Mapping Unicode to SQL character sets

For each character set *SQLCS* in the SQL-environment, there shall be an implementation-defined mapping *CSM* of strings of Unicode to strings of *SQLCS*.

4.10.3 Mapping SQL <identifier>s to XML

Since not every SQL <identifier> is an acceptable XML Name, it is necessary to define a mapping of SQL <identifier>s to XML Names. This mapping is defined in Subclause 9.1, “Mapping SQL <identifier>s to XML Names”. The basic idea of this mapping is that characters that are not valid in XML Names are converted to a sequence of hexadecimal digits derived from the Unicode encoding of the character, bracketed by an introductory underscore and lowercase `x` and a trailing underscore.

There are two variants of the mapping, known as *partially escaped* and *fully escaped*. The two differences are in the treatment of non-initial <colon> and the treatment of an <identifier> beginning with the letters `xml` in any combination of upper or lower case. The fully escaped variant maps a non-initial <colon> to **`_x003A_`**, whereas the partially escaped variant maps non-initial <colon> to **`:`**. Also, the fully escaped variant maps initial `xml` and `XML` to **`_x0078_ml`** and **`_x0058_ML`**, respectively, whereas the partially escaped does not.

NOTE 12 — This part of this International Standard specifies no syntax for invoking the partially escaped mapping specified in Subclause 9.1, “Mapping SQL <identifier>s to XML Names”. This specification is intended to be used by applications and referenced by other standards.

4.10.4 Mapping XML Names to SQL

A single algorithm suffices to reverse both the partially escaped and the fully escaped variants of the mapping of SQL <identifier>s to XML Names. This algorithm is found in Subclause 9.3, “Mapping XML Names to SQL <identifier>s”. The basic idea is to scan the XML Name from left to right, looking for escape sequences of the form **`_x`*`NNNN`*`_`** or **`_x`*`NNNNNN`*`_`** where ***`N`*** denotes a hexadecimal digit. Such sequences are converted to the character of SQL_TEXT that corresponds to the Unicode code point U+0000*NNNN* or U+00*NNNNNN*, respectively.

NOTE 13 — This part of this International Standard specifies no syntax for invoking the mapping specified in Subclause 9.3, “Mapping XML Names to SQL <identifier>s”. This specification is intended to be used by applications and referenced by other standards. It is the responsibility of any such application or other standard to ensure that the correct number of arguments as well as a valid value for each argument are supplied for this mapping.

NOTE 14 — The sequence of mappings from SQL <identifier> to XML Name (using either the fully escaped mapping or the partially escaped mapping) to SQL <identifier> restores the original SQL <identifier> (assuming that every character in the source SQL-implementation's SQL <identifier> is a character of SQL_TEXT in the target SQL-implementation). However, the sequence of mappings from XML Name to SQL <identifier> to XML Name does not necessarily restore the XML Name. Also, more than one XML Name may be mapped to the same SQL <identifier>.

4.10.5 Mapping SQL data types to XML

For each SQL type or domain that is not unmappable, there is a corresponding XML Schema type. The mapping is fully specified in Subclause 9.5, “Mapping SQL data types to XML Schema data types”. The following is a conceptual description of this mapping.

In general, each SQL predefined type, distinct type, or domain *SQLT* is mapped to the XML Schema type ***`XMLT`*** that is the closest analog to *SQLT*. Since the value space of ***`XMLT`*** is frequently richer than the set of values that can be represented by *SQLT*, facets are used to restrict ***`XMLT`*** in order to capture the restrictions on *SQLT* as much as possible.

In addition, many of the distinctions in the SQL type system (for example, CHARACTER VARYING *versus* CHARACTER LARGE OBJECT) have no corresponding distinction in the XML Schema type system. In order to represent these distinctions, XML Schema annotations are defined. The content of the annotations is defined by this standard; however, whether such annotations are actually generated is implementation-defined. Elements from the XML namespace identified by the XML namespace prefix "**`sqlxml`**" are used to populate these annotations.

The SQL character string types are mapped to the XML Schema type **`xs:string`**. For the SQL type CHARACTER, if the mapping of the SQL character set to Unicode is homomorphic, then fixed length strings are mapped to fixed length strings, and the facet **`xs:length`** is used. Otherwise (*i.e.*, CHARACTER when the mapping is not homomorphic, as well as CHARACTER VARYING and CHARACTER LARGE OBJECT), the facet **`xs:maxLength`** is used. Annotations optionally indicate the precise SQL type (CHARACTER, CHARACTER VARYING, or CHARACTER LARGE OBJECT), the length or maximum length of the SQL type, the character set, and the default collation.

The SQL binary string types are mapped to either the XML Schema type **`xs:hexBinary`** or the XML Schema type **`xs:base64Binary`**. The **`xs:maxLength`** facet is set to the maximum length of the binary string in octets. Annotations optionally indicate the SQL type (BINARY LARGE OBJECT) and the maximum length in octets. For <XML element> and <XML forest>, the choice of whether to map to **`xs:hexBinary`** or **`xs:base64Binary`** is governed by the innermost <XML binary encoding> whose scope includes the <XML element> or <XML forest>; the default is implementation-defined. When mapping an SQL table, schema or catalog to XML, the choice is governed by a parameter, as specified in Subclause 9.11, "Mapping an SQL table to XML and an XML Schema document", Subclause 9.14, "Mapping an SQL schema to an XML document and an XML Schema document", and Subclause 9.17, "Mapping an SQL catalog to an XML document and an XML Schema document".

The exact numeric SQL types NUMERIC and DECIMAL are mapped to the XML Schema type **`xs:decimal`** using the facets **`xs:precision`** and **`xs:scale`**. It is implementation-defined whether the SQL types INTEGER, SMALLINT, and BIGINT are mapped to the XML Schema type **`xs:integer`** using the facets **`xs:maxInclusive`** and **`xs:minInclusive`** or to the closest XML Schema type that is a subtype of **`xs:integer`**, using the facets **`xs:maxInclusive`** and **`xs:minInclusive`** if the range of the SQL type does not exactly match the range of the XML Schema type to which it is mapped. Annotations optionally indicate the SQL type (NUMERIC, DECIMAL, INTEGER, SMALLINT, or BIGINT), precision of NUMERIC, user-specified precision of DECIMAL (which may be less than the actual precision), and scale of NUMERIC and DECIMAL.

The approximate numeric SQL types are mapped to either the XML Schema type **`xs:float`**, if the binary precision is less than or equal to 24 binary digits (bits) and the range of the binary exponent lies between -149 and 104, inclusive; otherwise, the XML Schema type **`xs:double`** is used. Annotations optionally indicate the SQL type (REAL, DOUBLE PRECISION, or FLOAT), the binary precision, the minimum and maximum values of the range of binary exponents, and, for FLOAT, the user-specified binary precision (which may be less than the actual precision).

The SQL type BOOLEAN is mapped to the XML Schema type **`xs:boolean`**. Optionally, an annotation indicates the SQL type (BOOLEAN).

The SQL type DATE is mapped to the XML Schema type **`xs:date`**. The **`xs:pattern`** facet is used to exclude the possibility of a time zone displacement. Optionally, an annotation indicates the SQL type, DATE.

The SQL types TIME WITHOUT TIME ZONE and TIME WITH TIME ZONE are mapped to the XML Schema type **`xs:time`**. The **`xs:pattern`** facet is used to exclude the possibility of a time zone displacement, in the case of TIME WITHOUT TIME ZONE, or to require a time zone displacement, in the case of TIME WITH TIME ZONE. The pattern also reflects the fractional seconds precision of the SQL type. Annotations optionally indicate the SQL type (TIME or TIME WITH TIME ZONE) and the fractional seconds precision.

The SQL types TIMESTAMP WITHOUT TIME ZONE and TIMESTAMP WITH TIME ZONE are mapped to the XML Schema type **`xs:dateTime`**. The **`xs:pattern`** facet is used to exclude the possibility of a time zone displacement, in the case of TIMESTAMP WITHOUT TIME ZONE, or to require a time zone displacement, in the case of TIMESTAMP WITH TIME ZONE. The pattern also reflects the fractional seconds precision of the SQL type. Annotations optionally indicate the SQL type (TIMESTAMP or TIMESTAMP WITH TIME ZONE) and the fractional seconds precision.

The SQL interval types are mapped to the XML Query types **`xs:yearMonthDuration`** and **`xs:dayTimeDuration`**. The **`xs:pattern`** facet is used to require precisely the year, month, day, hour, minute and second fields indicated by the SQL type. The pattern also reflects the leading field precision and the fractional seconds precision (when applicable). Annotations optionally indicate the SQL type, leading field precision and (when applicable) the fractional seconds precision.

An SQL row type is mapped to an XML Schema complex type that consists of one element for each field of the SQL row type. For each field *F* of the SQL row type, the name of the corresponding XML element is obtained by mapping the field name of *F* using the fully escaped variant, and the XML Schema type of the element is obtained by mapping the field type of *F*.

An SQL domain is mapped to XML by mapping the domain's data type to XML and then optionally applying to the generated XML Schema type an annotation that identifies the name of the domain.

An SQL distinct type is mapped to an XML Schema simple type by mapping the source type of the distinct type. Optionally, an annotation specifying the name of the distinct type is applied to the generated XML Schema type.

An SQL collection type is mapped to an XML Schema complex type having a single XML element named **`element`** whose XML Schema type is obtained by mapping the element type of the SQL collection type. This XML element is defined using **`minOccurs="0"`**. For an SQL array type, **`maxOccurs`** is the maximum cardinality of the array, whereas for an SQL multiset type, **`maxOccurs="unbounded"`**.

An SQL XML type is mapped to an XML Schema complex type that allows mixed content and an unvalidated **`any`** section. Optionally, an annotation indicates the SQL type, XML.

4.10.6 Mapping values of SQL data types to XML

For each SQL type or domain *SQLT*, with the exception of structured types and reference types, there is also a mapping of values of type *SQLT* to the value space of the corresponding XML Schema type. The mappings of values are largely determined by the data type mappings. The precise rules for non-null values are found in Subclause 9.8, “Mapping values of SQL data types to values of XML Schema data types”. The mappings for values of predefined types are designed to exploit <cast specification> as much as possible. As for null values, there is generally a choice of whether to represent nulls using absence or **`xsi:nil="true"`**. However, for elements of a collection type, null values are always represented by **`xsi:nil="true"`**.

4.10.7 Mapping XQuery atomic values to SQL values

As defined in [XQueryDM], an XQuery atomic type is either an XML Schema primitive type, or derived from an XML Schema primitive type by restriction (and not by union or list).

Let ***AV*** be an XQuery atomic value. Let ***AT*** be the XQuery atomic type of ***AV***. Let ***PT*** be given by

Case:

— If ***AT*** is an XML Schema primitive type, then ***AT***.

— If ***AT*** is **`xs:yearMonthDuration`**, or derived from **`xs:yearMonthDuration`**, then **`xs:yearMonthDuration`**.

— If ***AT*** is **`xs:dayTimeDuration`**, or derived from **`xs:dayTimeDuration`**, then **`xs:dayTimeDuration`**.

— Otherwise, the XML Schema primitive type from which ***AT*** is derived.

This part of ISO/IEC 9075 (notably, in Subclause 6.6, “<XML cast specification>”) regards ***AV*** as being a value belonging to some category of SQL predefined type, as follows.

Case:

— If ***PT*** is **`xs:string`**, then ***AV*** is regarded as being a character string whose character repertoire is Unicode.

— If ***PT*** is **`xs:hexBinary`** or **`xs:base64Binary`**, then ***AV*** is regarded as being a binary string.

— If ***PT*** is **`xs:decimal`**, then ***AV*** is regarded as being an exact numeric value.

— If ***PT*** is **`xs:float`** or **`xs:double`**, then ***AV*** is regarded as being an approximate numeric value.

— If ***PT*** is **`xs:time`** and the XQuery datetime timezone component of ***AV*** is an empty XQuery sequence, then the XQuery datetime normalized value of ***AV*** is regarded as being a value of type TIME WITHOUT TIME ZONE.

— If ***PT*** is **`xs:time`** and the XQuery datetime timezone component of ***AV*** is not an empty XQuery sequence, then ***AV*** is regarded as being a value of type TIME WITH TIME ZONE, in which the XQuery datetime timezone component of ***AV*** is the timezone component, and the XQuery datetime normalized value is the UTC component.

— If ***PT*** is **`xs:dateTime`**, the XQuery datetime normalized value ***XDNV*** of ***AV*** is positive, and the XQuery datetime timezone component of ***AV*** is an empty XQuery sequence, then ***XDNV*** is regarded as being a value of type TIMESTAMP WITHOUT TIME ZONE. If ***XDNV*** would have a SECOND field greater than or equal to 59 in a minute of UTC that has exactly 59 seconds, then it is implementation-defined whether an implementation-defined value of type TIMESTAMP WITHOUT TIME ZONE is identified with ***XDNV***, or whether an exception condition is raised: *data exception — datetime field overflow*.

— If ***PT*** is **`xs:dateTime`**, the XQuery datetime normalized value of ***AV*** is positive, and the XQuery datetime timezone component of ***AV*** is not an empty XQuery sequence, then ***AV*** is regarded as being a value of type TIMESTAMP WITH TIME ZONE, in which the XQuery datetime timezone component of ***AV*** is the timezone component, and the XQuery datetime normalized value is the UTC component. If AV denotes a minute of UTC that has exactly 59 seconds and the SECOND field ***AV*** is greater than or equal to 59, then it is implementation-defined whether an implementation-defined value of type TIMESTAMP WITHOUT TIME ZONE is identified with ***AV***, or whether an exception condition is raised: *data exception — datetime field overflow*.

— If ***PT*** is **`xs:date`**, the XQuery datetime normalized value of ***AV*** is positive, and the XQuery datetime timezone component of ***AV*** is an empty XQuery sequence, then the XQuery datetime normalized value of ***AV*** is regarded as being a value of type DATE.

— If ***PT*** is **`xs:yearMonthDuration`**, then ***AV*** is regarded as being a year-month interval.

— If ***PT*** is **`xs:dayTimeDuration`**, then ***AV*** is regarded as being a day-time interval.

— If ***PT*** is **`xs:boolean`**, then ***AV*** is regarded as being a value of type BOOLEAN.

4.10.8 Visibility of columns, tables, and schemas in mappings from SQL to XML

An *XML unmappable data type* is a data type that is one of the following: a structured type, a reference type, XML(SEQUENCE), or a type defined in some part of ISO/IEC 9075 other than [ISO9075-2] and this part. An *XML unmappable column* is a column that has a declared type that is one of the XML unmappable data types or has a declared type that is based on an XML unmappable data type.

A column *C* of table *T* is a *visible column* of *T* for authorization identifier *U* if the applicable privileges for *U* include the SELECT privilege on *C* and, if the declared type of *C* is a distinct type, the applicable privileges for *U* include EXECUTE on the user-defined cast function identified by the Syntax Rules of Subclause 6.5, “<cast specification>”. A column *C* of table *T* is an *XML visible column* of *T* for authorization identifier *U* if *C* is a visible column of *T* for authorization identifier *U* and the declared type of *C* is not an XML unmappable data type.

A table *T* of schema *S* is a *visible table* of *S* for authorization identifier *U* if *T* is either a base table or a viewed table that contains a column *C* that is a visible column for *U*. A table *T* of schema *S* is an *XML visible table* of *S* for authorization identifier *U* if *T* is either a base table or a viewed table that contains a column *C* that is an XML visible column for *U*.

A schema *S* of catalog *C* is a *visible schema* of *C* for authorization identifier *U* if *S* contains a table *T* that is a visible table for *U*. A schema *S* of catalog *C* is an *XML visible schema* of *C* for authorization identifier *U* if *S* contains a table *T* that is an XML visible table for *U*.

4.10.9 Mapping an SQL table to XML

Subclause 9.11, “Mapping an SQL table to XML and an XML Schema document”, defines a mapping of an SQL table to one or both of two documents: an XML Schema document that describes the structure of the mapped XML and either an XML document or a sequence of XML elements. Only base tables and viewed tables may be the source of this mapping.

NOTE 15 — This part of this International Standard specifies no syntax for invoking the mapping specified in Subclause 9.11, “Mapping an SQL table to XML and an XML Schema document”. This specification is intended to be used by applications and referenced by other standards. It is the responsibility of any such application or other standard to ensure that the correct number of arguments as well as a valid value for each argument are supplied for this mapping.

Only the XML visible columns of this table for the user that invokes this mapping will be represented in the generated XML.

This mapping allows the invoker to specify:

9.4 Mapping an SQL data type to an XML Name

Function

Define the mapping of an SQL data type or domain to an XML Name.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *D* be the SQL data type or the underlying data type of the domain provided for an application of this Subclause.
2) If *D* is a character string type, then:
   a) Let *SQLCS* be the character set of *D*.
   b) Let *N* be the length or maximum length of *D*.
   c) Let *CSM* be the implementation-defined mapping of strings of *SQLCS* to strings of Unicode. Let *MAXCSL* be the maximum length in characters of *CSM*(*S*), for all strings *S* of length *N* characters.
   d) Let ***`MLIT`*** be the canonical XML Schema literal of the XML Schema type **`xs:integer`** denoting *MAXCSL*.
   e) Let ***`NLIT`*** be the canonical XML Schema literal denoting *N* in the lexical representation of XML Schema type **`xs:integer`**.
   f) Case:
      i) If *CSM* is homomorphic, and *N* equals *MAXCSL*, then

         Case:

         1) If the type designator of *D* is CHARACTER, then let ***`XMLN`*** be the following:

            `CHAR_`***`MLIT`***

         2) If the type designator of *D* is CHARACTER VARYING, then let ***`XMLN`*** be the following:

            `VARCHAR_`***`MLIT`***

         3) If the type designator of *D* is CHARACTER LARGE OBJECT, then let ***`XMLN`*** be the following:

            `CLOB_`***`MLIT`***

ii) If *CSM* is homomorphic, and *N* does not equal *MAXCSL*, then

Case:

1) If the type designator of *D* is CHARACTER, then let ***XMLN*** be the following:

CHAR_*NLIT*_*MLIT*

2) If the type designator of *D* is CHARACTER VARYING, then let ***XMLN*** be the following:

VARCHAR_*NLIT*_*MLIT*

3) If the type designator of *D* is CHARACTER LARGE OBJECT, then let ***XMLN*** be the following:

CLOB_*NLIT*_*MLIT*

iii) Otherwise,

Case:

1) If the type designator of *D* is CHARACTER or CHARACTER VARYING, then let ***XMLN*** be the following:

VARCHAR_*NLIT*_*MLIT*

2) If the type designator of *D* is CHARACTER LARGE OBJECT, then let ***XMLN*** be the following:

CLOB_*NLIT*_*MLIT*

3) If the type designator of *D* is BINARY LARGE OBJECT, then:

a) Let *N* be the maximum length of *D*. Let ***XN*** be the canonical XML Schema literal denoting *N* in the lexical representation of XML Schema type **xs:integer**.

b) Let ***XMLN*** be the following:

BLOB_*XN*

4) If the type designator of *D* is NUMERIC, then:

a) Let *P* be the precision of *D*. Let ***XP*** be the canonical XML Schema literal denoting *P* in the lexical representation of XML Schema type **xs:integer**.

b) Let *S* be the scale of *D*. Let ***XS*** be the canonical XML Schema literal denoting *S* in the lexical representation of XML Schema type **xs:integer**.

c) Let ***XMLN*** be the following:

NUMERIC_*XP*_*XS*

5) If the type designator of *D* is DECIMAL, then:

a) Let *P* be the precision of *D*. Let ***XP*** be the canonical XML Schema literal denoting *P* in the lexical representation of XML Schema type **xs:integer**.

b) Let *S* be the scale of *D*. Let ***XS*** be the canonical XML Schema literal denoting *S* in the lexical representation of XML Schema type **`xs:integer`**.

c) Let ***XMLN*** be the following:

```
DECIMAL_XP_XS
```

6) If the type designator of *D* is INTEGER, then let ***XMLN*** be the following:

```
INTEGER
```

7) If the type designator of *D* is SMALLINT, then let ***XMLN*** be the following:

```
SMALLINT
```

8) If the type designator of *D* is BIGINT, then let ***XMLN*** be the following:

```
BIGINT
```

9) If the type designator of *D* is FLOAT, then:

a) Let *P* be the precision of *D*. Let ***XP*** be the canonical XML Schema literal denoting *P* in the lexical representation of XML Schema type **`xs:integer`**.

b) Let ***XMLN*** be the following:

```
FLOAT_XP
```

10) If the type designator of *D* is REAL, then let ***XMLN*** be the following:

```
REAL
```

11) If the type designator of *D* is DOUBLE PRECISION, then let ***XMLN*** be the following:

```
DOUBLE
```

12) If the type designator of *D* is BOOLEAN, then let ***XMLN*** be the following:

```
BOOLEAN
```

13) If the type designator of *D* is TIME WITHOUT TIME ZONE, then:

a) Let *TP* be the time precision of *D*. Let ***XTP*** be the canonical XML Schema literal denoting *TP* in the lexical representation of XML Schema type **`xs:integer`**.

b) Let ***XMLN*** be the following:

```
TIME_XTP
```

14) If the type designator of *D* is TIME WITH TIME ZONE, then:

a) Let *TP* be the time precision of *D*. Let ***XTP*** be the canonical XML Schema literal denoting *TP* in the lexical representation of XML Schema type **`xs:integer`**.

b) Let ***XMLN*** be the following:

```
TIME_WTZ_XTP
```

15) If the type designator of *D* is TIMESTAMP WITHOUT TIME ZONE, then:

    a) Let *TSP* be the timestamp precision of *D*. Let ***XTSP*** be the canonical XML Schema literal denoting *TSP* in the lexical representation of XML Schema type **`xs:integer`**.

    b) Let ***XMLN*** be the following:

    ```
    TIMESTAMP_XTSP
    ```

16) If the type designator of *D* is TIMESTAMP WITH TIME ZONE, then:

    a) Let *TSP* be the timestamp precision of *D*. Let ***XTSP*** be the canonical XML Schema literal denoting *TSP* in the lexical representation of XML Schema type **`xs:integer`**.

    b) Let ***XMLN*** be the following:

    ```
    TIMESTAMP_WTZ_XTSP
    ```

17) If the type designator of *D* is DATE, then let ***XMLN*** be the following:

    ```
    DATE
    ```

18) If *D* is a domain, then let *C*, *S*, and *N* be the catalog name, schema name, and domain name of *D*, respectively. Let ***XMLN*** be the result of applying the mapping defined in Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", to "Domain", *C*, *S*, and *N*.

19) If *D* is a row type, then let the XML Name ***IDI*** be an implementation-dependent identifier for the row type. Two row types that have different numbers of fields, different field names, or different declared types in corresponding fields shall have different values of ***IDI***. It is implementation-dependent whether the types of two sites of row type, having the same number of fields, and having corresponding fields of the same name and declared type, receive the same row type identifier. Let ***XMLN*** be **`Row.`*****`IDI`***.

20) If *D* is a distinct type, then let *C*, *S*, and *N* be the catalog name, schema name, and type name of *D*, respectively. Let ***XMLN*** be the result of applying the mapping defined in Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", to "UDT", *C*, *S*, and *N*.

21) If *D* is an array type, then let *ET* be the element type of *D* and let *M* be the maximum cardinality of *D*. Let ***XMLET*** be the result of applying this Subclause to *ET*. Let ***MLIT*** be the canonical XML Schema literal denoting *M* in the lexical representation of XML Schema type **`xs:integer`**. Let ***XMLN*** be **`Array_`*****`MLIT`*****`.`*****`XMLET`***.

22) If *D* is a multiset type, then let *ET* be the element type of *D*. Let ***XMLET*** be the result of applying this Subclause to *ET*. Let ***XMLN*** be **`Multiset.`*****`XMLET`***.

23) If *D* is an XML type, then let ***XMLN*** be **`XML`**.

24) ***XMLN*** is the XML Name that is result of this mapping.

Conformance Rules

None.

9.5 Mapping SQL data types to XML Schema data types

Function

Define the mapping of SQL data types and domains to XML Schema data types.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *SQLT* be *SQLTYPE*, the SQL data type or domain in an application of this Subclause.

2) Let *NULLS* be *NULLS*, the choice of whether to map null values to absent elements (absent) or to elements that are marked with **`xsi:nil="true"`** (nil).

   Case:

   a) If *NULLS* is absent, then let the XML text ***`XMLNULLS`*** be **`minOccurs="0"`**.

   b) If *NULLS* is nil, then let the XML text ***`XMLNULLS`*** be **`nillable="true"`**.

3) Let *ENCODING* be *ENCODING*, the choice of whether to encode binary strings in base64 or in hex.

4) Let *TM* be the implementation-defined mapping of character strings of SQL_TEXT to character strings of Unicode.

5) Let **`xs`** be the XML namespace prefix to be used to identify the XML Schema namespace as shown in Table 2, “XML namespace prefixes and their URIs”.

6) Let **`sqlxml`** be the XML namespace prefix to be used to identify the XML namespace as shown in Table 2, “XML namespace prefixes and their URIs”.

7) Let ***`XMLT`*** denote the representation of the XML Schema data type that is the mapping of *SQLT* into XML. ***`XMLT`*** is defined by the following rules.

8) Case:

   a) If *SQLT* is a character string type, then:

      i) Let *SQLCS* be the character set of *SQLT*. Let *SQLCSN* be the name of *SQLCS*. Let *N* be the length or maximum length of *SQLT*.

      ii) Let *CSM* be the implementation-defined mapping of strings of *SQLCS* to strings of Unicode. Let *MAXCSL* be the maximum length in characters of *CSM*(*S*), for all strings *S* of length *N* characters.

iii) Let ***NLIT*** and ***MLIT*** be canonical XML Schema literals of the XML Schema type `xs:integer` denoting *N* and *MAXCSL*, respectively.

iv) Case:

1) If the type designator of *SQLT* is CHARACTER, then:

A) Case:

I) If *CSM* is homomorphic, then let ***FACET*** be the XML text:

```
<xs:length value="MLIT">
```

II) Otherwise, let ***FACET*** be the XML text:

```
<xs:maxLength value="MLIT">
```

B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or given by:

```
name="CHAR"
```

C) It is implementation-defined whether the XML text ***ANNL*** is the zero-length string or given by:

```
length="NLIT"
```

2) If the type designator of *SQLT* is CHARACTER VARYING, then:

A) Let ***FACET*** be the XML text:

```
<xs:maxLength value="MLIT">
```

B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or given by:

```
name="VARCHAR"
```

C) It is implementation-defined whether the XML text ***ANNL*** is the zero-length string or given by:

```
maxLength="NLIT"
```

3) If the type designator of *SQLT* is CHARACTER LARGE OBJECT, then:

A) Let ***FACET*** be the XML text:

```
<xs:maxLength value="MLIT">
```

B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or given by:

```
name="CLOB"
```

C) It is implementation-defined whether the XML text ***ANNL*** is the zero-length string or given by:

```
maxLength="NLIT"
```

v) Let the XML text ***SQLCSNLIT*** be the result of mapping *SQLCSN* to Unicode using *TM*. It is implementation-defined whether the XML text ***ANNCS*** is the zero-length string or given by:

```
characterSetName="SQLCSNLIT"
```

vi) Let *SQLCON* be the name of the collation of *SQLT*. Let the XML text ***SQLCONLIT*** be the result of mapping *SQLCON* to Unicode using *TM*. It is implementation-defined whether the XML text ***ANNCO*** is the zero-length string or given by:

```
collation="SQLCONLIT"
```

vii) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or given by:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                    ANNT ANNL ANNCS ANNCO/>
  </xs:appinfo>
</xs:annotation>
```

viii) ***XMLT*** is the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  <xs:restriction base="xs:string">
    FACET
  </xs:restriction>
</xs:simpleType>
```

b) If *SQLT* is a binary string type, then:

i) Let *N* be the length or maximum length of *SQLT*. Let ***NLIT*** be an XML Schema literal denoting *N* in the lexical representation of the XML Schema type **xs:integer**.

ii) Case:

1) If *ENCODING* indicates that binary strings are to be encoded in hex, then let ***EN*** be the XML text **hexBinary**.

2) Otherwise, let ***EN*** be the XML text **base64Binary**.

iii) Let ***FACET*** be the XML text:

```
<xs:maxLength value="NLIT">
```

iv) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or given by:

```
name="BLOB"
```

v) It is implementation-defined whether the XML text ***ANNL*** is the zero-length string or given by:

```
maxLength="NLIT"
```

vi) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or given by:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                       ANNT ANNL/>
  </xs:appinfo>
</xs:annotation>
```

vii) ***XMLT*** is the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  <xs:restriction base="xs:EN">
    FACET
  </xs:restriction>
</xs:simpleType>
```

c) If the type designator of *SQLT* is NUMERIC or DECIMAL, then:

i) Let *P* be the precision of *SQLT*. Let ***PLIT*** be an XML Schema literal denoting *P* in the lexical representation of the XML Schema type **xs:integer**. Let ***FACETP*** be the XML text:

```
<xs:totalDigits value="PLIT"/>
```

ii) Let *S* be the scale of *SQLT*. Let ***SLIT*** be an XML Schema literal denoting *S* in the lexical representation of the XML Schema type **xs:integer**. Let ***FACETS*** be the XML text:

```
<xs:fractionDigits value="SLIT"/>
```

iii) Case:

1) If the type designator of *SQLT* is NUMERIC, then:

A) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or

```
name="NUMERIC"
```

B) It is implementation-defined whether the XML text ***ANNP*** is the zero-length string or:

```
precision="PLIT"
```

2) If the type designator of *SQLT* is DECIMAL, then:

A) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="DECIMAL"
```

B) Let *UP* be the value of the <precision> specified in the <data type> used to create the descriptor of *SQLT*. Let ***UPLIT*** be an XML Schema literal denoting *UP* in the lexical representation of the XML Schema type **xs:integer**. It is implementation-defined whether the XML text ***ANNP*** is the zero-length string or:

```
userPrecision="UPLIT"
```

NOTE 40 — *UP* may be less than *P*, as specified in SR 27) of Subclause 6.1, “<data type>”, in [ISO9075-2].

iv) It is implementation-defined whether the XML text ***ANNS*** is the zero-length string or:

```
scale="SLIT"
```

v) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                    ANNT ANNP ANNS/>
  </xs:appinfo>
</xs:annotation>
```

vi) ***XMLT*** is the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  <xs:restriction base="xs:decimal">
    FACETP
    FACETS
  </xs:restriction>
</xs:simpleType>
```

d) If the type designator of *SQLT* is INTEGER, SMALLINT, or BIGINT, then:

i) Let *MAX* be the maximum value representable by *SQLT*. Let ***MAXLIT*** be an XML Schema literal denoting *MAX* in the lexical representation of the XML Schema type **`xs:integer`**. Let ***FACETMAX*** be the XML text:

```
<xs:maxInclusive value="MAXLIT"/>
```

ii) Let *MIN* be the minimum value representable by *SQLT*. Let ***MINLIT*** be an XML Schema literal denoting *MIN* in the lexical representation of the XML Schema type **`xs:integer`**. Let ***FACETMIN*** be the XML text:

```
<xs:minInclusive value="MINLIT"/>
```

iii) Case:

1) If the type designator of *SQLT* is INTEGER, then it is implementation-defined whether the XML text ***ANN*** is the zero-length string or:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                    name="INTEGER"/>
  </xs:appinfo>
</xs:annotation>
```

2) If the type designator of *SQLT* is SMALLINT, then it is implementation-defined whether the XML text ***ANN*** is the zero-length string or:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                       name="SMALLINT"/>
  </xs:appinfo>
</xs:annotation>
```

3) If the type designator of *SQLT* is BIGINT, then it is implementation-defined whether the XML text ***ANN*** is the zero-length string or:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                    name="BIGINT"/>
  </xs:appinfo>
</xs:annotation>
```

iv) It is implementation-defined whether *REST* is

```
<xs:restriction base="xs:integer">
  FACETMAX
  FACETMIN
</xs:restriction>
```

or determined by

1) Case:

A) If there is no row in Table 3, “Constraining facets of XML Schema integer types” such that *MAX* is less than or equal to the value in the “maxInclusive” column and *MIN* is greater than or equal to the value in the “minInclusive” column, then:

I) Let ***TYPE*** be **`xs:integer`**.

II) Let ***FMAX*** be *FACETMAX*.

III) Let ***FMIN*** be *FACETMIN*.

B) Otherwise:

I) Let ***TYPE*** be the contents of the “Type” column, in Table 3, “Constraining facets of XML Schema integer types”, taken from the first row in the table for which *MAX* is less than or equal to the value in the “maxInclusive” column and *MIN* is greater than or equal to the value in the “minInclusive” column.

II) If *MAX* is equal to the value of the “maxInclusive”, in the selected row of the table, then let ***FMAX*** be the zero-length string; otherwise, let ***FMAX*** be ***FACETMAX***.

III) If *MIN* is equal to the value of the “minInclusive”, in the selected row of the table, then let ***FMIN*** be the zero-length string; otherwise, let ***FMIN*** be ***FACETMIN***.

2) Let *REST* be:

```
<xs:restriction base="TYPE">
```

```
  FMAX
  FMIN
</xs:restriction>
```

Table 3 — Constraining facets of XML Schema integer types

Type	minInclusive	maxInclusive
`xs:unsignedByte`	0	255
`xs:byte`	-128	127
`xs:unsigned-Short`	0	2^{16}-1 (65,535)
`xs:short`	-2^{15} (-32,768)	2^{15}-1 (32,767)
`xs:unsignedInt`	0	2^{32}-1 (4,294,967,295)
`xs:int`	-2^{31} (-2,147,483,648)	2^{31}-1 (2,147,483,647)
`xs:unsignedLong`	0	2^{64}-1 (18,446,744,073,709,551,615)
`xs:long`	-2^{63} (-9,223,372,036,854,775,808)	2^{63}-1 (9,223,372,036,854,775,807)

v) ***XMLT*** is the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  REST
</xs:simpleType>
```

e) If *SQLT* is approximate numeric, then:

i) Let *P* be the binary precision of *SQLT*, let *MINEXP* be the minimum binary exponent supported by *SQLT*, and let *MAXEXP* be the maximum binary exponent supported by *SQLT*.

ii) Case:

1) If *P* is less than or equal to 24 binary digits (bits), *MINEXP* is greater than or equal to -149, and *MAXEXP* is less than or equal to 104, then let the XML text ***TYPE*** be **`float`**.

2) Otherwise, let the XML text ***TYPE*** be **`double`**.

iii) Case:

1) If the type designator of *SQLT* is REAL, then the XML text ***ANNUP*** is the zero-length string, and it is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="REAL"
```

2) If the type designator of *SQLT* is DOUBLE PRECISION, then the XML text ***ANNUP*** is the zero-length string, and it is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="DOUBLE PRECISION"
```

3) Otherwise:

A) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="FLOAT"
```

B) Let *UP* be the value of the <precision> specified in the <data type> used to create the descriptor of *SQLT*. Let ***UPLIT*** be an XML Schema literal denoting *UP* in the lexical representation of the XML Schema type **xs:integer**. It is implementation-defined whether the XML text ***ANNUP*** is the zero-length string or:

```
userPrecision="UPLIT"
```

NOTE 41 — *UP* may be less than *P*, as specified in SR 29) of Subclause 6.1, "<data type>", in [ISO9075-2].

iv) Let ***PLIT*** be an XML Schema literal denoting *P* in the lexical representation of the XML Schema type **xs:integer**. It is implementation-defined whether the XML text ***ANNP*** is the zero-length string or:

```
precision="PLIT"
```

v) Let ***MINLIT*** be an XML Schema literal denoting *MINEXP* in the lexical representation of the XML Schema type **xs:integer**. It is implementation-defined whether the XML text ***ANNMIN*** is the zero-length string or:

```
minExponent="MINLIT"
```

vi) Let ***MAXLIT*** be an XML Schema literal denoting *MAXEXP* in the lexical representation of the XML Schema type **xs:integer**. It is implementation-defined whether the XML text ***ANNMAX*** is the zero-length string or:

```
maxExponent="MAXLIT"
```

vii) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                    ANNT ANNP ANNUP ANNMAX ANNMIN/>
  </xs:appinfo>
</xs:annotation>
```

viii) It is implementation-defined whether ***XMLT*** is **xs:*TYPE*** or the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  <xs:restriction base="xs:TYPE">
```

```
    </xs:restriction>
  </xs:simpleType>
```

f) If the type designator of *SQLT* is BOOLEAN, then it is implementation-defined whether ***XMLT*** is **xs:boolean** or the XML Schema type defined by:

```
<xs:simpleType>
  <xs:annotation>
    <xs:appinfo>
      <sqlxml:sqltype kind="PREDEFINED"
                      name="BOOLEAN"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:restriction base="xs:boolean"/>
</xs:simpleType>
```

g) If the type designator of *SQLT* is DATE, then:

i) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                    name="DATE"/>
  </xs:appinfo>
</xs:annotation>
```

ii) ***XMLT*** is the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  <xs:restriction base="xs:date">
    <xs:pattern
      value="\p{Nd}{4}-\p{Nd}{2}-\p{Nd}{2}"/>
  </xs:restriction>
</xs:simpleType>
```

h) If *SQLT* is TIME WITHOUT TIME ZONE, then:

i) Let *S* be the <time fractional seconds precision> of *SQLT*. Let ***SLIT*** be an XML Schema literal denoting *S* in the lexical representation of XML Schema type **xs:integer**.

ii) Case:

1) If *S* is greater than 0 (zero), then let the XML text ***FACETP*** be:

```
<xs:pattern value=
    "\p{Nd}{2}:\p{Nd}{2}:\p{Nd}{2}.\p{Nd}{SLIT}"/>
```

2) Otherwise, let the XML text ***FACETP*** be:

```
<xs:pattern value=
    "\p{Nd}{2}:\p{Nd}{2}:\p{Nd}{2}"/>
```

iii) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="TIME"
```

iv) It is implementation-defined whether the XML text ***ANNS*** is the zero-length string or:

```
scale="SLIT"
```

v) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                    ANNT ANNS/>
  </xs:appinfo>
</xs:annotation>
```

vi) ***XMLT*** is the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  <xs:restriction base="xs:time">
    FACETP
  </xs:restriction>
</xs:simpleType>
```

i) If *SQLT* is TIME WITH TIME ZONE, then:

i) Let *S* be the <time fractional seconds precision> of *SQLT*. Let ***SLIT*** be an XML Schema literal denoting *S* in the lexical representation of XML Schema type **xs:integer**.

ii) Let the XML text ***TZ*** be:

```
(+|-)\p{Nd}{2}:\p{Nd}{2}
```

iii) Case:

1) If *S* is greater than 0 (zero), then let the XML text ***FACETP*** be:

```
<xs:pattern value=
    "\p{Nd}{2}:\p{Nd}{2}:\p{Nd}{2}.\p{Nd}{SLIT}TZ"/>
```

2) Otherwise, let the XML text ***FACETP*** be:

```
<xs:pattern value=
    "\p{Nd}{2}:\p{Nd}{2}:\p{Nd}{2}TZ"/>
```

iv) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="TIME WITH TIME ZONE"
```

v) It is implementation-defined whether the XML text ***ANNS*** is the zero-length string or:

```
scale="SLIT"
```

vi) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                    ANNT ANNS/>
  </xs:appinfo>
</xs:annotation>
```

vii) ***XMLT*** is the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  <xs:restriction base="xs:time">
    FACETP
  </xs:restriction>
</xs:simpleType>
```

j) If *SQLT* is TIMESTAMP WITHOUT TIME ZONE, then:

i) Let *S* be the <time fractional seconds precision> of *SQLT*. Let ***SLIT*** be an XML Schema literal denoting *S* in the lexical representation of XML Schema type `xs:integer`.

ii) Let the XML text ***DATETIME*** be:

```
\p{Nd}{4}-\p{Nd}{2}-\p{Nd}{2}T\p{Nd}{2}:\p{Nd}{2}:\p{Nd}{2}
```

iii) Case:

1) If *S* is greater than 0 (zero), then let the XML text ***FACETP*** be:

```
<xs:pattern value=
    "DATETIME.\p{Nd}{SLIT}"/>
```

2) Otherwise, let the XML text ***FACETP*** be:

```
<xs:pattern value=
    "DATETIME"/>
```

iv) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="TIMESTAMP"
```

v) It is implementation-defined whether the XML text ***ANNS*** is the zero-length string or:

```
scale="SLIT"
```

vi) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                    ANNT ANNS/>
  </xs:appinfo>
</xs:annotation>
```

vii) ***XMLT*** is the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  <xs:restriction base="xs:dateTime">
    FACETP
  </xs:restriction>
</xs:simpleType>
```

k) If *SQLT* is TIMESTAMP WITH TIME ZONE, then:

i) Let *S* be the <time fractional seconds precision> of *SQLT*. Let ***SLIT*** be an XML Schema literal denoting *S* in the lexical representation of XML Schema type `xs:integer`.

ii) Let the XML text ***DATETIME*** be:

```
\p{Nd}{4}-\p{Nd}{2}-\p{Nd}{2}T\p{Nd}{2}:\p{Nd}{2}:\p{Nd}{2}
```

iii) Let the XML text ***TZ*** be:

```
(+|-)\p{Nd}{2}:\p{Nd}{2}
```

iv) Case:

1) If *S* is greater than 0 (zero), then let the XML text ***FACETP*** be:

```
<xs:pattern value=
    "DATETIME.\p{Nd}{SLIT}TZ"/>
```

2) Otherwise, let the XML text ***FACETP*** be:

```
<xs:pattern value="DATETIMETZ"/>
```

v) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="TIMESTAMP WITH TIME ZONE"
```

vi) It is implementation-defined whether the XML text ***ANNS*** is the zero-length string or:

```
scale="SLIT"
```

vii) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                    ANNT ANNS/>
  </xs:appinfo>
</xs:annotation>
```

viii) ***XMLT*** is the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  <xs:restriction base="xs:dateTime">
    FACETP
  </xs:restriction>
</xs:simpleType>
```

l) If the type designator of *SQLT* is INTERVAL, then:

i) Let *P* be the <interval leading field precision> of *SQLT*. Let ***PLIT*** be an XML Schema literal for *P* in the XML Schema type **xs:integer**. It is implementation-defined whether the XML text ***ANNP*** is the zero-length string or:

```
leadingPrecision="PLIT"
```

ii) Case:

1) If the <end field> or <single datetime field> of *SQLT* specifies SECOND, then let *S* be the <interval fractional seconds precision> of *SQLT*, and let ***SLIT*** be an XML Schema literal for *S* in the XML Schema type **xs:integer**. Let the XML text ***SECS*** be:

```
\p{Nd}{2}.\p{Nd}{SLIT}S
```

It is implementation-defined whether the XML text ***ANNS*** is the zero-length string or:

```
scale="SLIT"
```

2) Otherwise, let the XML text ***ANNS*** be the zero-length string, and let the XML text ***SECS*** be:

```
\p{Nd}{2}S
```

iii) Case:

1) If *SQLT* is INTERVAL YEAR then:

A) Let the XML text ***FACETP*** be:

```
<xs:pattern value="-?P\p{Nd}{PLIT}Y"/>
```

B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="INTERVAL YEAR"
```

2) If *SQLT* is INTERVAL YEAR TO MONTH then:

A) Let the XML text ***FACETP*** be:

```
<xs:pattern value="-?P\p{Nd}{PLIT}Y\p{Nd}{2}M"/>
```

B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="INTERVAL YEAR TO MONTH"
```

3) If *SQLT* is INTERVAL MONTH then:

A) Let the XML text ***FACETP*** be:

```
<xs:pattern value="-?P\p{Nd}{PLIT}M"/>
```

B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="INTERVAL MONTH"
```

4) If SQLT is INTERVAL DAY then:

A) Let the XML text ***FACETP*** be:

```
<xs:pattern value="-?P\p{Nd}{PLIT}D"/>
```

B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="INTERVAL DAY"
```

5) If *SQLT* is INTERVAL DAY TO HOUR then:

A) Let the XML text ***FACETP*** be:

```
<xs:pattern value="-?P\p{Nd}{PLIT}DT\p{Nd}{2}H"/>
```

B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="INTERVAL DAY TO HOUR"
```

6) If SQLT is INTERVAL DAY TO MINUTE then:

A) Let the XML text ***FACETP*** be:

```
<xs:pattern value=
  "-?P\p{Nd}{PLIT}DT\p{Nd}{2}H\p{Nd}{2}M"/>
```

B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="INTERVAL DAY TO MINUTE"
```

7) If *SQLT* is INTERVAL DAY TO SECOND then:

A) Let the XML text ***FACETP*** be:

```
<xs:pattern value=
  "-?P\p{Nd}{PLIT}DT\p{Nd}{2}H\p{Nd}{2}MSECS"/>
```

B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="INTERVAL DAY TO SECOND"
```

8) If SQLT is INTERVAL HOUR then:

A) Let the XML text ***FACETP*** be:

```
<xs:pattern value="-?PT\p{Nd}{PLIT}H"/>
```

B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="INTERVAL HOUR"
```

9) If *SQLT* is INTERVAL HOUR TO MINUTE then:

   A) Let the XML text ***FACETP*** be:

   ```
   <xs:pattern value=
     "-?PT\p{Nd}{PLIT}H\p{Nd}{2}M"/>
   ```

   B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

   ```
   name="INTERVAL HOUR TO MINUTE"
   ```

10) If *SQLT* is INTERVAL HOUR TO SECOND then:

   A) Let the XML text ***FACETP*** be:

   ```
   <xs:pattern value=
     "-?PT\p{Nd}{PLIT}H\p{Nd}{2}MSECS"/>
   ```

   B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

   ```
   name="INTERVAL HOUR TO SECOND"
   ```

11) If *SQLT* is INTERVAL MINUTE then:

   A) Let the XML text ***FACETP*** be:

   ```
   <xs:pattern value="-?PT\p{Nd}{PLIT}M"/>
   ```

   B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

   ```
   name="INTERVAL MINUTE"
   ```

12) If *SQLT* is INTERVAL MINUTE TO SECOND then:

   A) Let the XML text ***FACETP*** be:

   ```
   <xs:pattern value="-?PT\p{Nd}{PLIT}MSECS"/>
   ```

   B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

   ```
   name="INTERVAL MINUTE TO SECOND"
   ```

13) If *SQLT* is INTERVAL SECOND then:

   A) Let the XML text ***FACETP*** be:

   ```
   <xs:pattern value="-?PTSECS"/>
   ```

   B) It is implementation-defined whether the XML text ***ANNT*** is the zero-length string or:

   ```
   name="INTERVAL SECOND"
   ```

iv) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or:

```
<xs:annotation>
```

```
        <xs:appinfo>
          <sqlxml:sqltype kind="PREDEFINED"
                            ANNT ANNP ANNS/>
        </xs:appinfo>
      </xs:annotation>
```

v) Case:

1) If *SQLT* is a year-month duration, then let ***DTYPE*** be **xs:yearMonthDuration**.

2) If *SQLT* is a day-time duration, then let ***DTYPE*** be **xs:dayTimeDuration**.

vi) ***XMLT*** is the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  <xs:restriction base="DTYPE">
    FACETP
  </xs:restriction>
</xs:simpleType>
```

m) If *SQLT* is a domain, then:

i) Let *DT* be the data type of *SQLT*.

ii) Let ***XMLN*** be the XML Name obtained by applying Subclause 9.4, “Mapping an SQL data type to an XML Name”, to *DT*.

iii) Let *DC*, *DS*, and *DN* be the domain's catalog name, schema name, and domain name, respectively.

iv) Let *DCLIT*, *DSLIT*, and *DNLIT* be the result of mapping *DC*, *DS*, and *DN* to Unicode using *TM*.

v) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or given by:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="DOMAIN"
                    catalogName="DCLIT" schemaName="DSLIT"
                    typeName="DNLIT" mappedType="XMLN"/>
  </xs:appinfo>
</xs:annotation>
```

vi) ***XMLT*** is the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  <xs:restriction base="XMLN"/>
</xs:simpleType>
```

n) If *SQLT* is a row type, then:

i) Let *N* be the number of fields of *SQLT*. Let FT_i and FN_i be the declared type and name of the *i*-th field of *SQLT*, respectively, for *i* between 1 (one) and *N*.

ii) Let $\boldsymbol{XMLMT_i}$ be the result of applying the mapping in Subclause 9.4, “Mapping an SQL data type to an XML Name”, to FT_i, for *i* between 1 (one) and *N*.

iii) For each *i* between 1 (one) and *N*, the General Rules of Subclause 9.1, “Mapping SQL <identifier>s to XML Names”, are applied with FN_i as *IDENT*, using the fully escaped variant of the mapping, resulting in $\mathbf{XMLFN}_i$.

iv) Let $\mathbf{FNLIT}_i$ be the result of mapping FN_i to Unicode using *TM*, for *i* between 1 (one) and *N*.

v) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or given by:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="ROW">
      <sqlxml:field name="FNLIT1"
                    mappedType="XMLMT1"/>
      . . .
      <sqlxml:field name="FNLITN"
                    mappedType="XMLMTN"/>
    </sqlxml:sqltype>
  </xs:appinfo>
</xs:annotation>
```

vi) ***XMLT*** is the XML Schema type defined by:

```
<xs:complexType>
  ANN
  <xs:sequence>
    <xs:element name="XMLFN1" type="XMLMT1" XMLNULLS/>
    . . .
    <xs:element name="XMLFNN" type="XMLMTN" XMLNULLS/>
  </xs:sequence>
</xs:complexType>
```

o) If *SQLT* is a distinct type, then:

i) Let *ST* be the source type of *SQLT*.

ii) Let ***XMLN*** be the XML Name obtained by applying Subclause 9.4, “Mapping an SQL data type to an XML Name”, to *ST*.

iii) Let *DTC*, *DTS*, and *DTN* be the catalog name, schema name, and type name, respectively, of *SQLT*.

iv) Let *DTCLIT*, *DTSLIT*, and *DTNLIT* be the result of mapping *DTC*, *DTS*, and *DTN* to Unicode using *TM*.

v) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or given by:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="DISTINCT"
                    catalogName="DTCLIT" schemaName="DTSLIT"
                    typeName="DTNLIT" mappedType="XMLN"
                    final="true"/>
  </xs:appinfo>
</xs:annotation>
```

vi) ***XMLT*** is the XML Schema type defined by:

```
<xs:simpleType>
  ANN
  <xs:restriction base="XMLN"/>
</xs:simpleType>
```

p) If *SQLT* is an array type, then:

i) Let *ET* be the element type of *SQLT*, and let *M* be the maximum cardinality of *SQLT*.

ii) Let ***XMLN*** be the XML Name obtained by applying Subclause 9.4, “Mapping an SQL data type to an XML Name”, to *ET*.

iii) Let ***MLIT*** be an XML Schema literal for *M* in the XML Schema type **xs:integer**.

iv) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or given by:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="ARRAY"
                    maxElements="MLIT"
                    mappedElementType="XMLN"/>
  </xs:appinfo>
</xs:annotation>
```

v) ***XMLT*** is the XML Schema type defined by:

```
<xs:complexType>
  ANN
  <xs:sequence>
    <xs:element name="element" minOccurs="0"
                maxOccurs="MLIT" nillable="true"
                type="XMLN"/>
  </xs:sequence>
</xs:complexType>
```

q) If *SQLT* is a multiset type, then:

i) Let *ET* be the element type of *SQLT*.

ii) Let ***XMLN*** be the XML Name obtained by applying Subclause 9.4, “Mapping an SQL data type to an XML Name”, to *ET*.

iii) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or given by:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="MULTISET"
                    mappedElementType="XMLN"/>
  </xs:appinfo>
</xs:annotation>
```

iv) ***XMLT*** is the XML Schema type defined by:

```
<xs:complexType>
  ANN
```

```
    <xs:sequence>
      <xs:element name="element" minOccurs="0"
                   maxOccurs="unbounded" nillable="true"
                   type="XMLN"/>
    </xs:sequence>
  </xs:complexType>
```

r) If *SQLT* is an XML type other than XML(SEQUENCE), then:

   i) It is implementation-defined whether the XML text ***ANN*** is the zero-length string or:

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                    name="XML"/>
  </xs:appinfo>
</xs:annotation>
```

   ii) ***XMLT*** is the XML Schema type defined by:

```
<xs:complexType mixed="true">
  ANN
  <xs:sequence>
    <xs:any name="element" minOccurs="0" maxOccurs="unbounded"
             processContents="skip"/>
  </xs:sequence>
</xs:complexType>
```

9) ***XMLT*** is the result of this mapping.

Conformance Rules

None.

9.6 Mapping an SQL data type to a named XML Schema data type

Function

Define the mapping of an SQL data type or domain to an XML Schema data type.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *D* be the SQL data type or domain provided for an application of this subclause.

**** Editor's Note (number 2) ****

The following data types have no mappings: structured types, and reference types. Note that both metadata and data mappings are required. See Language Opportunity **XML-019** in the Editor's Notes.

2) Let ***`XMLN`*** be the result of applying the mapping defined in Subclause 9.4, "Mapping an SQL data type to an XML Name", to *D*.

3) Let *NULLS* be the choice of whether to map null values to absent elements (absent) or to elements that are marked with **`xsi:nil="true"`** (nil).

4) Let *ENCODING* be the choice of whether to encode binary strings in base64 or in hex.

5) If *D* is a character string type, then:

   a) Let *SQLCS* be the character set of *D*.

   b) Let *N* be the length or maximum length of *D*.

   c) Let *CSM* be the implementation-defined mapping of strings of *SQLCS* to strings of Unicode. Let *MAXCSL* be the maximum length in characters of *CSM*(*S*), for all strings *S* of length *N* characters.

   d) Let ***`MLIT`*** be the canonical XML Schema literal of the XML Schema type **`xs:integer`** denoting *MAXCSL*.

   e) Case:

      i) If *CSM* is homomorphic, *N* equals *MAXCSL*, and the type designator of *D* is CHARACTER, then let ***`SQLCDT`*** be the following:

```
<xs:simpleType name="XMLN">
   <xs:restriction base="xs:string">
      <xs:length value="MLIT" />
   </xs:restriction>
</xs:simpleType>
```

ii) Otherwise, let ***SQLCDT*** be the following:

```
<xs:simpleType name="XMLN">
   <xs:restriction base="xs:string">
      <xs:maxLength value="MLIT" />
   </xs:restriction>
</xs:simpleType>
```

6) If *D* is a domain or a data type that is not a character string type, then:

a) Let *ENC* be the choice of whether to encode binary strings in base64 or in hex, let *NC* be the choice of whether to map null values to absent elements or elements that are marked with **xsi:nil="true"**, and let ***XMLT*** be the XML Schema data type that is the result of applying the mapping defined in Subclause 9.5, “Mapping SQL data types to XML Schema data types”, with *D* as *SQLTYPE*, *ENC* as *ENCODING*, and *NC* as *NULLS*.

b) Case:

i) If *D* is an XML type, then *XMLT* is of the form

```
<xs:complexType MIXED>
  XMLTC
</xs:complexType>
```

where *XMLTC* is the string comprising the element content and *MIXED* is the string comprising the attribute of the element. Let *SQLDT* be the following:

```
<xs:complexType name="XMLN" MIXED>
  XMLTC
</xs:complexType>
```

ii) If ***XMLT*** is of the form **<xs:complexType>*XMLTC*</xs:complexType>**, where ***XMLTC*** is the string comprising the element content, then let ***SQLCDT*** be the following:

```
<xs:complexType name="XMLN">
   XMLTC
</xs:complexType>
```

iii) If ***XMLT*** is of the form **<xs:simpleType>*XMLTC*</xs:simpleType>**, where ***XMLTC*** is the string comprising the element content, then let ***SQLCDT*** be the following:

```
<xs:simpleType name="XMLN">
   XMLTC
</xs:simpleType>
```

iv) Otherwise, let ***SQLCDT*** be the following:

```
<xs:simpleType name="XMLN">
   <xs:restriction base="XMLT" />
</xs:simpleType>
```

7) ***SQLCDT*** is the XML Schema data type that is the result of this mapping.

Conformance Rules

None.

9.7 Mapping a collection of SQL data types to XML Schema data types

Function

Define the mapping of a collection of SQL data types and domains to XML Schema data types.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *NULLS* be the choice of whether to map null values to absent elements (absent), or whether to map them to elements that are marked with `xsi:nil="true"` (nil).

2) Let *ENCODING* be the choice of whether to encode binary strings in base64 or in hex.

3) Let *C* be the collection of SQL data types and domains provided for an application of this Subclause. *C* is augmented recursively as follows, until no more data types are added to *C*:

   a) If *DO* is a domain contained in *C* and the data type of *DO* is not contained in *C*, then the data type of *DO* is added to *C*.

   b) If *RT* is a row type contained in *C* and *F* is a field of *RT* whose declared type is not contained in *C*, then the declared type of *F* is added to *C*.

   c) If *DT* is a distinct type contained in *C* whose source type is not in *C*, then the source type of *DT* is added to *C*.

   d) If *CT* is a collection type contained in *C* whose element type is not in *C*, then the element type of *CT* is added to *C*.

4) Let *n* be the number of SQL data types and domains in *C*.

5) Let ***`XMLD`*** be the zero-length string. Let ***`XMLTL`*** be an empty list of XML Names.

6) For *i* ranging from 1 (one) to *n*:

   a) Let D_i be the *i*-th SQL data type or domain in *C*.

   b) Let $\mathbf{XMLN}_i$ be the result of applying the mapping defined in Subclause 9.4, "Mapping an SQL data type to an XML Name", to D_i.

   c) Let $\mathbf{XMLT}_i$ be the XML Schema data type that is the result of applying the mapping defined in Subclause 9.6, "Mapping an SQL data type to a named XML Schema data type", to D_i using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with `xsi:nil="true"` and *ENCODING* as the choice of whether to encode binary strings in base64 or in hex.

d) Two XML Names are considered to be equivalent to each other if they have the same number of characters and the Unicode values of all corresponding characters are equal.

e) If ***XMLN$_i$*** is not equivalent to the value of any XML Name in ***XMLTL***, then:

i) Let ***XMLD*** be:

```
XMLD || XMLTi
```

ii) Append ***XMLN$_i$*** to ***XMLTL***.

7) ***XMLD*** contains the XML Schema data types that are the result of this mapping.

Conformance Rules

None.

9.8 Mapping values of SQL data types to values of XML Schema data types

Function

Define the mapping of non-null values of SQL data types to XML.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *DV* be *the value of an SQL data type* in an application of this Subclause.

   Case:

   a) If *DV* is a value of a distinct type, then let *SQLT* be the source type of the distinct type, and let *SQLV* be the result of:

   ```
   CAST (DV AS SQLT)
   ```

   b) Otherwise, let *SQLT* be the most specific type of *DV* and let *SQLV* be *DV*.

2) Let *NULLS* be the choice of whether to map null values to absent elements (absent) or to elements that are marked with **`xsi:nil="true"`** (nil).

3) Let *ENCODING* be the choice of whether to encode binary strings in base64 or in hex.

4) Let *CHARMAPPING* be the choice of whether to replace certain characters (“&” (U+0026), “<” (U+003C), “>” (U+003E), and Carriage Return (U+000D)) with their character references (*True*) or not (*False*).

5) Let *NL* be the choice of whether to map null values to absent elements or to elements that are marked with **`xsi:nil="true"`**, let *ENC* be the choice of whether to encode binary strings in base64 or in hex, and let ***`XMLT`*** be the XML Schema data type that is the result of applying the mapping defined in Subclause 9.5, “Mapping SQL data types to XML Schema data types”, to *SQLT* as *SQLTYPE*, *NL* as *NULLS* and *ENC* as *ENCODING*.

6) Let *M* be the implementation-defined maximum length of variable-length character strings.

7) If *SQLT* is not a binary string type, a character string type, a row type, a collection type, an interval type, or the XML type, then let *CV* be the result of

   ```
   CAST ( SQLV AS CHARACTER VARYING(M) )
   ```

8) Let *CSM* be the implementation-defined mapping of the default character set of CHARACTER VARYING to Unicode.

9) Case:

a) If *SQLT* is a character string type, then:

i) Let *CS* be the character set of *SQLT*. Let ***XMLVRAW*** be the XML text that is the result of mapping *SQLV* to Unicode using the implementation-defined mapping of character strings of *CS* to Unicode.

ii) Case:

1) If the SQL-implementation supports Feature X211, “XML 1.1 support”, then if any Unicode code point in ***XMLVRAW*** does not represent a valid XML 1.1 character, then an exception condition is raised: *SQL/XML mapping error — invalid XML character*.

2) Otherwise, if any Unicode code point in ***XMLVRAW*** does not represent a valid XML 1.0 character, then an exception condition is raised: *SQL/XML mapping error — invalid XML character*.

iii) Case:

1) If *CHARMAPPING* is <u>True</u>, then let ***XMLV*** be ***XMLVRAW***, with each instance of “&” (U+0026) replaced by “&amp;”, each instance of “<” (U+003C) replaced by “&lt;”, each instance of “>” (U+003E) replaced by “&gt;”, and each instance of Carriage Return (U+000D) replaced by “&#x0d;”.

2) Otherwise, let ***XMLV*** be ***XMLVRAW***.

b) If *SQLT* is a binary string type, then

Case:

i) If *ENCODING* indicates that binary strings are to be encoded in hex, then let ***XMLV*** be the hex encoding as defined by [Schema2] of *SQLV*.

ii) Otherwise, let ***XMLV*** be the base64 encoding as defined by [Schema2] of *SQLV*.

c) If *SQLT* is a numeric type, then let ***XMLV*** be the result of mapping *CV* to Unicode using *CSM*.

d) If *SQLT* is a BOOLEAN, then let *TEMP* be the result of:

```
LOWER (CV)
```

Let ***XMLV*** be the result of mapping *TEMP* to Unicode using *CSM*.

e) If *SQLT* is DATE, then let *TEMP* be the result of:

```
SUBSTRING (CV FROM 6 FOR 10)
```

Let ***XMLV*** be the result of mapping *TEMP* to Unicode using *CSM*.

f) If *SQLT* specifies TIME, then:

i) Let *P* be the <time fractional seconds precision> of *SQLT*.

ii) If *P* is 0 (zero), then let *Q* be 0 (zero); otherwise, let *Q* be *P* + 1 (one).

iii) If *SQLT* specifies WITH TIME ZONE, then let *Z* be 6; otherwise, let *Z* be 0 (zero).

iv) Case:

1) If the SECOND field of *CV* is greater than or equal to 60, then it is implementation-defined whether an implementation-defined value is assigned to *TEMP*, or whether to raise an exception condition: *data exception — datetime field overflow*.

2) Otherwise, let *TEMP* be the result of:

```
SUBSTRING (CV FROM 6 FOR 8 + Q + Z)
```

v) Let ***XMLV*** be the result of mapping *TEMP* to Unicode using *CSM*.

g) If *SQLT* specifies TIMESTAMP, then:

i) Let *P* be the <timestamp fractional seconds precision> of *SQLT*.

ii) If *P* is 0 (zero), then let *Q* be 0 (zero); otherwise, let *Q* be *P* + 1 (one).

iii) If *SQLT* specifies WITH TIME ZONE, then let *Z* be 6; otherwise, let *Z* be 0 (zero).

iv) Case:

1) If the SECOND field of *CV* is greater than or equal to 60, then it is implementation-defined whether an implementation-defined value is assigned to *TEMP*, or whether to raise an exception condition: *data exception — datetime field overflow*.

2) Otherwise, let *TEMP* be the result of:

```
   SUBSTRING (CV FROM 11 FOR 10)
|| 'T'
|| SUBSTRING (CV FROM 22 FOR 8 + Q + Z)
```

v) Let ***XMLV*** be the result of mapping *TEMP* to Unicode using *CSM*.

h) If *SQLT* specifies INTERVAL, then:

i) If *SQLV* is negative, then let *SIGN* be `'-'` (a character string of length 1 (one) consisting of <minus sign>); otherwise, let *SIGN* be the zero-length string.

ii) Let *SQLVA* be ABS(*SQLV*).

iii) Let *CVA* be the result of:

```
CAST ( SQLVA AS CHARACTER VARYING(M) )
```

iv) Let *L* be the <interval leading field precision> of *SQLT*.

v) Let *P* be the <interval fractional seconds precision> of *SQLT*, if any.

vi) If *P* is 0 (zero), then let *Q* be 0 (zero); otherwise, let *Q* be *P* + 1 (one).

vii) Case:

1) If *SQLT* is INTERVAL YEAR, then let *TEMP* be the result of:

```
SIGN || 'P' || SUBSTRING (CVA FROM 10 FOR L) || 'Y'
```

2) If *SQLT* is INTERVAL YEAR TO MONTH, then let *TEMP* be the result of

```
   SIGN || 'P'
```

```
|| SUBSTRING (CVA FROM 10 FOR L) || 'Y'
|| SUBSTRING (CVA FROM 11 + L FOR 2) || 'M'
```

3) If *SQLT* is INTERVAL MONTH, then let *TEMP* be the result of:

```
   SIGN || 'P'
|| SUBSTRING (CVA FROM 10 FOR L) || 'M'
```

4) If *SQLT* is INTERVAL DAY, then let *TEMP* be the result of:

```
   SIGN || 'P'
|| SUBSTRING (CVA FROM 10 FOR L) || 'D'
```

5) If *SQLT* is INTERVAL DAY TO HOUR, then let *TEMP* be the result of:

```
   SIGN || 'P'
|| SUBSTRING (CVA FROM 10 FOR L) || 'DT'
|| SUBSTRING (CVA FROM 11 + L FOR 2) || 'H'
```

6) If *SQLT* is INTERVAL DAY TO MINUTE, then let *TEMP* be the result of:

```
   SIGN || 'P'
|| SUBSTRING (CVA FROM 10 FOR L) || 'DT'
|| SUBSTRING (CVA FROM 11 + L FOR 2) || 'H'
|| SUBSTRING (CVA FROM 14 + L FOR 2) || 'M'
```

7) If *SQLT* is INTERVAL DAY TO SECOND, then let *TEMP* be the result of:

```
   SIGN || 'P'
|| SUBSTRING (CVA FROM 10 FOR L) || 'DT'
|| SUBSTRING (CVA FROM 11 + L FOR 2) || 'H'
|| SUBSTRING (CVA FROM 14 + L FOR 2) || 'M'
|| SUBSTRING (CVA FROM 17 + L FOR 2 + Q) || 'S'
```

8) If *SQLT* is INTERVAL HOUR, then let *TEMP* be the result of:

```
   SIGN || 'PT'
|| SUBSTRING (CVA FROM 10 FOR L) || 'H'
```

9) If *SQLT* is INTERVAL HOUR TO MINUTE, then let *TEMP* be the result of:

```
   SIGN || 'PT'
|| SUBSTRING (CVA FROM 10 FOR L) || 'H'
|| SUBSTRING (CVA FROM 11 + L FOR 2) || 'M'
```

10) If *SQLT* is INTERVAL HOUR TO SECOND, then let *TEMP* be the result of:

```
   SIGN || 'PT'
|| SUBSTRING (CVA FROM 10 FOR L) || 'H'
|| SUBSTRING (CVA FROM 11 + L FOR 2) || 'M'
|| SUBSTRING (CVA FROM 14 + L FOR 2 + Q) || 'S'
```

11) If *SQLT* is INTERVAL MINUTE, then let *TEMP* be the result of:

```
   SIGN || 'PT'
|| SUBSTRING (CVA FROM 10 FOR L) || 'M'
```

12) If *SQLT* is INTERVAL MINUTE TO SECOND, then let *TEMP* be the result of:

```
   SIGN || 'PT'
|| SUBSTRING (CVA FROM 10 FOR L) || 'M'
|| SUBSTRING (CVA FROM 11 + L FOR 2 + Q) || 'S'
```

13) If *SQLT* is INTERVAL SECOND, then let *TEMP* be the result of:

```
   SIGN || 'PT'
|| SUBSTRING (CVA FROM 10 FOR L + Q) || 'S'
```

viii) Let ***XMLV*** be the result of mapping *TEMP* to Unicode using *CSM*.

i) If *SQLT* is a row type, then:

i) Let *N* be the number of fields of *SQLT*. For *i* between 1 (one) and *N*, let FT_i, FN_i, and FV_i be the declared type, name, and value of the *i*-th field, respectively.

ii) For each *i* between 1 (one) and *N*:

1) The General Rules of Subclause 9.1, “Mapping SQL <identifier>s to XML Names”, are applied with FN_i as *IDENT*, using the fully escaped variant of the mapping, resulting in $\boldsymbol{XMLFN_i}$.

2) Case:

A) If FV_i is the null value and *NULLS* is absent, then let $\boldsymbol{XMLE_i}$ be the empty string.

B) If FV_i is the null value and *NULLS* is nil, then let $\boldsymbol{XMLE_i}$ be the XML element:

```
<XMLFNi xsi:nil="true"/>
```

C) Otherwise, let the XML text $\boldsymbol{XMLV_i}$ be the result of applying the mapping defined in this Subclause to FV_i. Let $\boldsymbol{XMLE_i}$ be the XML element:

```
<XMLFNi>XMLVi</XMLFNi>
```

iii) Let ***XMLV*** be:

$\boldsymbol{XMLE_1}$ $\boldsymbol{XMLE_2}$ . . . $\boldsymbol{XMLE_N}$

j) If *SQLV* is an array value or a multiset value, then:

i) Let *N* be the number of elements in *SQLV*.

ii) Let *ET* be the element type of *SQLV*.

iii) For *i* between 1 (one) and *N*:

1) Let E_i be the value of the *i*-th element of *SQLV*. (If *SQLV* is a multiset value, then the ordering of the elements is implementation-dependent.)

2) Case:

A) If E_i is the null value, then let the XML text ***XMLE*** be **`<element xsi:nil="true"/>`**.

B) Otherwise, let ***X_i*** be the result of applying this Subclause to E_i. Let the XML text ***$XMLE_i$*** be:

```
<element>Xi</element>
```

iv) Let ***XMLV*** be:

```
XMLE1 XMLE2 ... XMLEN
```

k) If *SQLT* is an XML type, then let ***XMLV*** be the serialized value of the XML value in *SQLV*:

```
XMLSERIALIZE (CONTENT SQLV AS CLOB)
```

10) ***XMLV*** is the result of this mapping.

Conformance Rules

None.

22 The SQL/XML XML Schema

22.1 The SQL/XML XML Schema

Function

Define the contents of the XML Schema for SQL/XML.

Syntax Rules

1) The contents of the SQL/XML XML Schema are:

```
<?xml version="1.0"?>
<xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://standards.iso.org/iso/9075/2003/sqlxml"
      xmlns:sqlxml="http://standards.iso.org/iso/9075/2003/sqlxml">
  <xs:annotation>
    <xs:documentation>
      This document contains definitions of types and
      annotations as specified in ISO/IEC 9075-14.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleType name="kindKeyword">
    <xs:restriction base="xs:string">
      <xs:enumeration value="PREDEFINED"/>
      <xs:enumeration value="DOMAIN"/>
      <xs:enumeration value="ROW"/>
      <xs:enumeration value="DISTINCT"/>
      <xs:enumeration value="ARRAY"/>
      <xs:enumeration value="MULTISET"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="typeKeyword">
    <xs:restriction base="xs:string">
      <xs:enumeration value="CHAR"/>
      <xs:enumeration value="VARCHAR"/>
      <xs:enumeration value="CLOB"/>
      <xs:enumeration value="BLOB"/>
      <xs:enumeration value="NUMERIC"/>
      <xs:enumeration value="DECIMAL"/>
      <xs:enumeration value="INTEGER"/>
      <xs:enumeration value="SMALLINT"/>
      <xs:enumeration value="BIGINT"/>
      <xs:enumeration value="FLOAT"/>
      <xs:enumeration value="REAL"/>
      <xs:enumeration value="DOUBLE PRECISION"/>
      <xs:enumeration value="BOOLEAN"/>
```

```
        <xs:enumeration value="DATE"/>
        <xs:enumeration value="TIME"/>
        <xs:enumeration value="TIME WITH TIME ZONE"/>
        <xs:enumeration value="TIMESTAMP"/>
        <xs:enumeration value="TIMESTAMP WITH TIME ZONE"/>
        <xs:enumeration value="INTERVAL YEAR"/>
        <xs:enumeration value="INTERVAL YEAR TO MONTH"/>
        <xs:enumeration value="INTERVAL MONTH"/>
        <xs:enumeration value="INTERVAL DAY"/>
        <xs:enumeration value="INTERVAL DAY TO HOUR"/>
        <xs:enumeration value="INTERVAL DAY TO MINUTE"/>
        <xs:enumeration value="INTERVAL DAY TO SECOND"/>
        <xs:enumeration value="INTERVAL HOUR"/>
        <xs:enumeration value="INTERVAL HOUR TO MINUTE"/>
        <xs:enumeration value="INTERVAL HOUR TO SECOND"/>
        <xs:enumeration value="INTERVAL MINUTE"/>
        <xs:enumeration value="INTERVAL MINUTE TO SECOND"/>
        <xs:enumeration value="INTERVAL SECOND"/>
        <xs:enumeration value="XML"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="fieldType">
      <xs:attribute name="name" type="xs:string"/>
      <xs:attribute name="mappedType" type="xs:string"/>
    </xs:complexType>
    <xs:element name="sqltype">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="field" type="sqlxml:fieldType"
                      minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="kind"
                      type="sqlxml:kindKeyword"/>
        <xs:attribute name="name"
                      type="sqlxml:typeKeyword" use="optional"/>
        <xs:attribute name="length" type="xs:integer"
                      use="optional"/>
        <xs:attribute name="maxLength" type="xs:integer"
                      use="optional"/>
        <xs:attribute name="characterSetName" type="xs:string"
                      use="optional"/>
        <xs:attribute name="collation" type="xs:string"
                      use="optional"/>
        <xs:attribute name="precision" type="xs:integer"
                      use="optional"/>
        <xs:attribute name="scale" type="xs:integer"
                      use="optional"/>
        <xs:attribute name="maxExponent" type="xs:integer"
                      use="optional"/>
        <xs:attribute name="minExponent" type="xs:integer"
                      use="optional"/>
        <xs:attribute name="userPrecision" type="xs:integer"
                      use="optional"/>
        <xs:attribute name="leadingPrecision" type="xs:integer"
                      use="optional"/>
        <xs:attribute name="maxElements" type="xs:integer"
                      use="optional"/>
        <xs:attribute name="catalogName" type="xs:string"
                      use="optional"/>
        <xs:attribute name="schemaName" type="xs:string"
                      use="optional"/>
        <xs:attribute name="domainName" type="xs:string"
                      use="optional"/>
        <xs:attribute name="typeName" type="xs:string"
                      use="optional"/>
        <xs:attribute name="mappedType" type="xs:string"
                      use="optional"/>
        <xs:attribute name="mappedElementType" type="xs:string"
                      use="optional"/>
        <xs:attribute name="final" type="xs:boolean"
                      use="optional"/>
      </xs:complexType>
    </xs:element>
    <xs:simpleType name="objectType">
       <xs:restriction base="xs:string">
          <xs:enumeration value="CATALOG" />
          <xs:enumeration value="SCHEMA" />
          <xs:enumeration value="BASE TABLE" />
          <xs:enumeration value="VIEWED TABLE" />
          <xs:enumeration value="CHARACTER SET" />
          <xs:enumeration value="COLLATION" />
       </xs:restriction>
    </xs:simpleType>
    <xs:element name="sqlname">
       <xs:complexType>
          <xs:attribute name="type" type="sqlxml:objectType"
                        use="required" />
          <xs:attribute name="catalogName" type="xs:string" />
          <xs:attribute name="schemaName" type="xs:string" />
          <xs:attribute name="localName" type="xs:string" />
       </xs:complexType>
    </xs:element>
  </xs:schema>
```

General Rules

None.

Conformance Rules

None.