WG3:DRS-020
H2-2002-365
August, 2002

ISO

International Organization for Standardization

ANSI

American National Standards Institute

ANSI TC NCITS H2

ISO/IEC JTC 1/SC 32/WG 3

Database

Title: (ISO-ANSI Working Draft) XML-Related Specifications (SQL/XML)

Author: Jim Melton (Editor)

References:

1) WG3:DRS-012 = H2-2002-357, *WD 9075-1 (SQL/Framework)*, August, 2002

2) WG3:DRS-013 = H2-2002-358, *WD 9075-2 (SQL/Foundation)*, August, 2002

3) WG3:DRS-014 = H2-2002-359, *WD 9075-3 (SQL/CLI)*, August, 2002

4) WG3:DRS-015 = H2-2002-360, *WD 9075-4 (SQL/PSM)*, August, 2002

5) WG3:DRS-016 = H2-2002-361, *WD 9075-9 (SQL/MED)*, August, 2002

6) WG3:DRS-017 = H2-2002-362, *WD 9075-10 (SQL/OLB)*, August, 2002

7) WG3:DRS-018 = H2-2002-363, *WD 9075-11 (SQL/Schemata)*, August, 2002

8) WG3:DRS-019 = H2-2002-364, *WD 9075-13 (SQL/JRT)*, August, 2002

9) WG3:DRS-020 = H2-2002-365, *WD 9075-14 (SQL/XML)*, August, 2002

ISO/IEC JTC 1/SC 32

Date: 2002-08-09

ISO/IEC 9075-14:200x(E)

ISO/IEC JTC 1/SC 32/WG 3

Secretariat: United States of America (ANSI)

Information technology — Database languages — SQL — Part 14: XML-Related Specifications (SQL/XML)

Technologies de l'information — Langages de base de donnée — SQL — Partie 14: «Specifications à XML» (SQL/XML)

Document type: International standard
Document subtype:
Document stage: (20) Working Draft
Document language: E

Contents

Page

TABLES

Tables **Page**

Foreword

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

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

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

Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 9075-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

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

— Part 1: Framework (SQL/Framework)

— Part 2: Foundation (SQL/Foundation)

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

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

— Part 9: Management of External Data (SQL/MED)

— Part 10: Object Language Bindings (SQL/OLB)

— Part 11: Information and Definition Schemas (SQL/Schemata)

— Part 13: SQL Routines and Types Using the Java Programming Language (SQL/JRT)

— Part 14: XML-Related Specifications (SQL/XML)

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

Introduction

The organization of this Part of this International Standard is as follows:

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

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

3) Clause 3, “Definitions, notations and conventions”, defines the notations and conventions used in this part of ISO/IEC 9075.

4) Clause 4, “Concepts”, presents concepts related to this part of ISO/IEC 9075.

5) Clause 5, “Lexical elements”, defines the lexical elements of the language.

6) Clause 6, “Scalar expressions”, defines the elements of the language that produce scalar values.

7) Clause 7, “Mappings”, defines the ways in which certain SQL information can be mapped into XML and certain XML information can be mapped into SQL.

8) Clause 8, “Additional common rules”, specifies the rules for assignments that retrieve data from or store data into SQL-data, and formation rules for set operations.

9) Clause 9, “Additional common elements”, defines additional language elements that are used in various parts of the language.

10) Clause 10, “SQL-client modules”, defines SQL-client modules and externally-invoked procedures.

11) Clause 11, “Dynamic SQL”, defines the SQL dynamic statements.

12) Clause 12, “The SQL/XML XML schema”, defines the content of an XML namespace that is used when SQL and XML are utilized together.

13) Clause 13, “Definition Schema”, defines base tables on which the viewed tables containing schema information depend.

14) Clause 14, “Status codes”, defines values that identify the status of the execution of SQL-statements and the mechanisms by which those values are returned.

15) Clause 15, “Conformance”, specifies the way in which conformance to this part of ISO/IEC 9075 may be claimed.

16) Annex A, “SQL Conformance Summary”, is an informative Annex. It summarizes the conformance requirements of the SQL language.

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

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

19) Annex D, “Incompatibilities with ISO/IEC 9075-2:1999”, is an informative Annex. It lists incompatibilities with the previous version of this part of ISO/IEC 9075.

20) Annex E, “SQL feature and package taxonomy”, is an informative Annex. It identifies features and packages of the SQL language specified in this part of ISO/IEC 9075 by an identifier and a short descriptive name. This taxonomy is used to specify conformance to the packages specified in this part of ISO/IEC 9075. The feature taxonomy may be used to develop other profiles involving the SQL language.

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

Information technology — Database languages — SQL —

Part 14: XML-Related Specifications (SQL/XML)

1 Scope

This part of ISO/IEC 9075 defines ways in which Database Language SQL can be used in conjunction with XML.

2 Normative references

The following normative documents contain provisions that, through reference in this text, constitute provisions of this International Standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

2.1 ISO/IEC JTC1 standards

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

ISO/IEC 9075-1:1999, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*.

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

ISO/IEC 10646-1:2000, *Information technology — Universal Multi-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*.

ISO/IEC FDIS 10646-2:2000, *Information technology — Universal Multi-Octet Coded Character Set (UCS) — Part 2: Supplementary Planes*.

2.2 Publicly-available specifications

The Unicode Consortium, *The Unicode Standard, Version 3.0*, Reading, MA, Addison-Wesley Developers Press,2000. ISBN 0-201-61633-5.

The Unicode Consortium, *The Unicode Standard, Version 3.1.0, Unicode Standard Annex #27: Unicode 3.1, (which amends The Unicode Standard, Version 3.0)*, 2001-03-23
`http://www.unicode.org/unicode/reports/tr27`

Davis, Mark and Dürst, Martin, *Unicode Standard Annex #15: Unicode Normalization Forms, Version 21.0*, 2001-03-23, The Unicode Consortium
`http://www.unicode.org/unicode/reports/tr15-21`

Davis, Mark, *Unicode Standard Annex #19: UTF-32, Version 8.0*, 2001-03-23, The Unicode Consortium
`http://www.unicode.org/unicode/reports/tr19-8`

WG3:DRS-020 = H2-2002-365

2.2 Publicly-available specifications

Extensible Markup Language (XML) Version 1.0 (second edition), 2 October, 2000, http://www.w3.org/TR/REC-xml

XML Path Language (XPath) Version 1.0, 16 November, 1999, http://www.w3.org/TR/xpath

Namespaces in XML, 14 January, 1999, http://www.w3.org/TR/REC-xml-names

(Recommendation) XML Schema Part 1: Structures, 2 May, 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

(Recommendation) XML Schema Part 2: Datatypes, 2 May, 2001, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

(Recommendation) Canonical XML Version 1.0, 15 March, 2001, http://www.w3.org/TR/xml-c14n

(Working Draft) XQuery 1.0: An XML Query Language, December 20, 2001, http://www.w3.org/TR/2001/WD-xquery-20011220/

(Working Draft) XML Path Language (XPath) 2.0, December 20, 2001, http://www.w3.org/TR/2001/WD-xpath20-20011220/

(Recommendation) XML Information Set, 24 October, 2001, http://www.w3.org/TR/2001/REC-xml-infoset-20011024

(Note) Unicode in XML and Other Markup Languages, 15 December, 2000, http://www.w3.org/TR/unicode-xml/

**Editor's Note **

Many of the normative references have not been finalized. It will be necessary to reference the correct specifications as they become available. This may entail changes to other clauses of this standard to align with the final forms of these specifications. See Possible Problem **XML-002** in the Editor's Notes.

3 Definitions, notations and conventions

This Clause modifies Clause 3, "Definitions, notations, and conventions", in ISO/IEC 9075-2.

3.1 Definitions

This Subclause modifies Subclause 3.1, "Definitions", in ISO/IEC 9075-2.

- 1 paragraph deleted.

3.1.1 Definitions provided in Part 14

This part of ISO/IEC 9075 defines the following terms:

a) **empty XML value**: the content of an XML element such as <a>, when the type of <a> is an XML complex type as defined by *XML Schema Part 2: Datatypes*.

b) **named expression**: an expression that has an alias specified with it, as seen in <XML element>, <XML forest>, <XML concatenation>, and <XML gen>.

c) **XML attribute**: an attribute as defined by *Extensible Markup Language (XML) Version 1.0 (second edition)*.

d) **XML content**: content as defined by *Extensible Markup Language (XML) Version 1.0 (second edition)*.

e) **XML document with prolog**: A document, as defined in *Extensible Markup Language (XML) Version 1.0 (second edition)*, rule 1, with a prolog consisting of at least an XMLDecl, as defined in *Extensible Markup Language (XML) Version 1.0 (second edition)*, rule 23.

f) **XML element**: an element as defined by *Extensible Markup Language (XML) Version 1.0 (second edition)*.

g) **XML forest of elements**: an ordered collection of zero or more XML elements.

h) **XQuery empty sequence**: an empty sequence as defined by *XQuery 1.0: An XML Query Language*.

i) **XQuery execution context**: an execution context as defined by *XQuery 1.0: An XML Query Language*.

j) **XQuery value**: a value as defined by *XQuery 1.0: An XML Query Language*.

k) **XQuery variable**: a variable as defined by *XQuery 1.0: An XML Query Language*.

3.1.2 Definitions taken from XML

This part of ISO/IEC 9075 makes use of the following terms defined in XML:

a) **DTD**

b) **NameChar**

c) **Name**

d) **URI**

e) **XML document**

f) **well-formed**

g) **well-formedness constraint**

3.2 Notations

This Subclause modifies Subclause 3.2, "Notation", in ISO/IEC 9075-2.

Insert this paragraph XML text, when represented in a conventional English-language paragraph, including Rules, is indicated using bold monospace font, for example, **<xsd:element>**. However, XML text that is presented on a separate line, as opposed to being incorporated in an English-language paragraph, and labeled as being XML text in an accompanying paragraph is written in monospace font (but not in boldface). For example:

```
<xsd:element>
```

Insert this paragraph Similarly, when a textual variable in a Rule denotes XML text, then the textual variable is written in italicized bold monospace font, for example, ***xsd***, and when the same textual variable appears in a separate line that is clearly marked as XML text, the textual variable is italicized but not bold.

Insert this paragraph Whenever XML text is presented, an implementation may substitute equivalent XML text, for example, through insertion or deletion insignificant blanks or new lines.

4 Concepts

This Clause modifies Clause 4, "Concepts", in ISO/IEC 9075-2.

4.1 Data types

This Subclause modifies Subclause 4.1, "Data types", in ISO/IEC 9075-2.

Replace the 5th paragraph Every predefined data type is a subtype of itself and of no other data types. It follows that every predefined data type is a supertype of itself and of no other data types. The predefined data types are individually described in each of Subclause 4.2, "Character strings", through Subclause 4.6, "Datetimes and intervals", in ISO/IEC 9075-2, and in Subclause 4.2, “XML”, in this part of ISO/IEC 9075.

Replace the 7th paragraph SQL defines predefined data types named by the following <key word>s: CHARACTER, CHARACTER VARYING, CHARACTER LARGE OBJECT, BINARY LARGE OBJECT, NUMERIC, DECIMAL, SMALLINT, INTEGER, BIGINT, FLOAT, REAL, DOUBLE PRECISION, BOOLEAN, DATE, TIME, TIMESTAMP, INTERVAL, and XML. These names are used in the *type designators* that constitute the *type precedence lists* specified in Subclause 9.5, "Type precedence list determination".

Augment the 8th paragraph

— The data type XML is referred to as the *XML type*. Values of the XML type are called *XML values*.

Augment the 20th paragraph

— A type *T* is *XML-ordered* if *T* is *X*-ordered, where *X* is the set consisting of the XML type.

4.2 XML

An XML data type is described by an XML data type descriptor. An XML data type descriptor contains:

— The name of the data type (XML).

4.2.1 Characteristics of XML values

Every value of XML type belongs to one of the following mutually exclusive categories:

— The null value.

— An XML document with prolog.

— An XML content.

Table 1, “Concatenation of two XML values”, shows the result of concatenation of two XML values, *X* followed by *Y*.

Table 1—Concatenation of two XML values

XML value *X*	**XML value *Y***	**Concatenation of *X* and *Y***
any XML value	*(null)*	*X*
(null)	any XML value	*Y*
XML document	XML document	*(error, not allowed)*
XML document	XML content	*(error, not allowed)*
XML content	XML document	*(error, not allowed)*
XML content	XML content	the XML content consisting of *X* followed by *Y*

4.2.2 XML comparison and assignment

Values corresponding to the data type XML are mutually assignable.

XML values are not comparable.

4.2.3 Operations involving XML values

<XML element> is an operator that returns an XML element given an XML element name, an optional list of XML attributes, and an optional list of values as the content of the new element. The values can be any value that has a mapping to an XML value.

<XML forest> is an operator that returns an XML forest of elements given a list of named expressions. An XML element is produced from each named expression, using the name as the XML element name and the value of the expression as the element content. The expressions can be any value that has a mapping to an XML value.

<XML gen> is an operator that returns an XML value by applying named expressions to an XQuery constructor. The expression names are mapped to XQuery variables used in the XQuery constructor. The expressions can be any value that has a mapping to an XML value.

<XML concat> is an operator that returns an XML forest of elements by concatenating a list of XML values, as defined in Subclause 4.2.1, “Characteristics of XML values”, Table 1, “Concatenation of two XML values”.

4.3 Data conversions

This Subclause modifies Subclause 4.11, "Data conversions", in ISO/IEC 9075-2.

Replace 1st paragraph Explicit data conversions can be specified by a <cast specification>. A <cast specification> defines how values of a source data type are converted into a value of a target data type according to the Syntax Rules and General Rules of Subclause 6.12, "<cast specification>", in ISO/IEC 9075-2 and Subclause 6.2, “<cast specification>”, in this part of ISO/IEC 9075. Data conversions between predefined data types and between constructed types are defined in ISO/IEC 9075-2. Data conversions between a user-defined type and another data type are defined by a user-defined cast.

4.4 Aggregate functions

This Subclause modifies Subclause 4.15.3, "Aggregate functions", in ISO/IEC 9075-2.

Add to the bulleted list following the 7th paragraph

— If XMLAGG is specified, then an XML forest of elements containing each element from the <value expression> evaluated for each row that qualifies.

4.5 Namespaces

This standard references certain namespaces that are defined by the World-Wide Web Consortium or by this standard. Each namespace is referenced using a namespace prefix. The namespace prefixes and their definitions are shown in Table 2, “Namespace prefixes and their URIs”.

Table 2—Namespace prefixes and their URIs

Namespace prefix	Namespace URI
xsd:	http://www.w3.org/2001/XMLSchema
xsi:	http://www.w3.org/2001/XMLSchema-instance
sqlxml:	http://www.iso-standards.org/mra/9075/2001/12/sqlxml

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

Editor’s Note
The value of the **sqlxml:** namespace identifier may change. See Possible Problem **XML-001**.

4.6 Mappings

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

— Mapping SQL character sets to XML character sets.

— 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 SQL data values to XML data values.

— 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.

4.6.1 Mapping SQL character sets to Unicode

For each character set *SQLCS* in the SQL-environment, there shall be an implementation-defined mapping *CSM* of strings of *SQLCS* to strings of Unicode. 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. Thus a homomorphic mapping has the property that fixed-length strings in *SQLCS* may be mapped to fixed-length strings in Unicode.

NOTE 1 – The XML entities **<**, **&**, **>**, **'**, and **"** are regarded as each representing a single character in XML, and do not pose an obstacle to defining homomorphic mappings.

Since SQL_TEXT is a character set in the SQL-environment, there shall be an implementation-defined mapping of strings of SQL_TEXT to Unicode. This mapping is called the *plain text mapping* from SQL to XML, and it is used to represent SQL text strings in XML when their use in XML is not as an XML Name (for example, in annotations).

4.6.2 Mapping SQL <identifier>s to XML names

Since not every SQL <identifier> is an acceptable XML Name, it is also necessary to define a mapping of SQL <identifier>s to XML Names. This mapping is defined in Subclause 7.1, "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 actually 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** (in any case combination) by prefixing **_xFFFF_**, whereas the partially escaped does not.

4.6.3 Mapping SQL data types to XML schema data types

For each SQL type or domain, there is defined a corresponding XML Schema type. The mapping is fully specified in Subclause 7.15, "Mapping SQL data types to XML schema data types". The following is a conceptual description of this mapping.

In general, each SQL predefined type or domain *SQLT* is mapped to the XML Schema built-in 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*, XML 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-dependent.

The SQL character string types are mapped to the XML Schema type **xsd:string**. For the SQL type CHARACTER, if the mapping of the SQL character set to the XML character set is homomorphic, then fixed length strings are mapped to fixed length strings, and the facet **xsd:length** is used. Otherwise (*i.e.*, CHARACTER when the mapping is not homomorphic, as well as CHARACTER VARYING and CHARACTER LARGE OBJECT), the facet **xsd: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 **xsd:hexBinary** or the XML Schema type **xsd:base64Binary**. It is implementation-dependent which of these types is used to map the SQL binary string types. The **xsd: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.

The exact numeric SQL types NUMERIC and DECIMAL are mapped to the XML Schema type **xsd:decimal** using the facets **xsd:precision** and **xsd:scale**. The SQL types INTEGER, SMALLINT, and BIGINT are mapped to the XML Schema type **xsd:integer** using the facets **xsd:maxInclusive** and **xsd:minInclusive**. 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 **xsd: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 **xsd: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 **xsd:boolean**. Optionally, an annotation indicates the SQL type (BOOLEAN).

The SQL type DATE is mapped to the XML Schema type **xsd:date**. The **xsd:pattern** facet is used to exclude the possibility of a time zone. 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 **xsd:time**. The **xsd:pattern** facet is used to exclude the possibility of a time zone, in the case of TIME WITHOUT TIME ZONE, or to require a time zone, 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 **xsd:dateTime**. The **xsd:pattern** facet is used to exclude the possibility of a time zone, in the case of TIMESTAMP WITHOUT TIME ZONE, or to require a time zone, 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 Schema type `xsd:duration`. The `xsd: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 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 distinct type is mapped to an XML Schema simple type by mapping the source type of the distinct type.

An SQL collection type is mapped to an XML Schema complex type having a single element named `element` whose XML Schema type is obtained by mapping the element type of the SQL collection type. This 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"`.

4.6.4 Mapping SQL data values to XML data values

For each SQL type or domain *SQLT*, 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 nonnull values are found in Subclause 7.16, “Mapping SQL data values to XML”. 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'`.

4.6.5 Visibility of columns, tables, and schemas in XML mappings

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*.

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 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*.

4.6.6 Mapping an SQL table to an XML document and an XML schema document

Subclause 7.3, “Mapping an SQL table to an XML document and an XML schema document”, defines a mapping between an SQL table and two documents, an XML document that reflects the data in the table, and an XML Schema document that describes the first document. Only base tables and viewed tables may be the source of this mapping.

These XML documents may be physical documents or virtual documents, depending upon the environment in which they are used.

Only the visible columns of this table for the user that invokes this mapping will be reflected in these two XML documents.

This mapping allows the invoker to specify whether to map null values to absent elements (absent), or whether to map them to elements that are marked with `xsi:nil="true"` (nil).

Some of the XML Schema type definitions and element definitions may contain annotation elements to reflect SQL metadata that is not directly relevant to XML. It is implementation-dependant whether these annotation elements are generated.

4.6.7 Mapping an SQL schema to an XML document and an XML schema document

Subclause 7.4, "Mapping an SQL schema to an XML document and an XML schema document", defines a mapping between the tables of an SQL schema and two documents, an XML document that reflects the data in these tables, and an XML Schema document that describes the first. These XML documents may be physical documents or virtual documents, depending upon the environment in which they are used.

Only the visible tables of the schema for the user that invokes this mapping will be reflected in these two XML documents. Only the visible columns of these tables for the user that invokes this mapping will be reflected in these two XML documents.

This mapping allows the invoker of this mapping to specify whether to map null values to absent elements (absent), or whether to map them to elements that are marked with `xsi:nil="true"` (nil).

Some of the XML Schema type definitions and element definitions may contain annotation elements to reflect SQL metadata that is not directly relevant to XML. It is implementation-dependant whether these annotation elements are generated.

4.6.8 Mapping an SQL catalog to an XML document and an XML schema document

Subclause 7.5, "Mapping an SQL catalog to an XML document and an XML schema document", defines a mapping between the tables of an SQL catalog and two documents, an XML document that reflects the data in these tables, and an XML Schema document that describes the first document. These XML documents may be physical documents or virtual documents, depending upon the environment in which they are used.

Only the visible schemas of this catalog for the user that invokes this mapping will be reflected in these two XML documents. Only the visible tables of these schemas for the user that invokes this mapping will be reflected in these two XML documents. Only the visible columns of these tables for the user that invokes this mapping will be reflected in these two XML documents.

This mapping allows the user that invokes this mapping to specify whether to map null values to absent elements (absent), or whether to map them to elements that are marked with `xsi:nil="true"` (nil).

Some of the XML Schema type definitions and element definitions may contain annotation elements to reflect SQL metadata that is not directly relevant to XML. It is implementation-dependant whether these annotation elements are generated.

4.6.9 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*.

4.6.10 Mapping XML Names to SQL <identifier>s

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 7.17, "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***NNNNNNNN***_ 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+*NNNNNNNN*, respectively.

NOTE 2 – The sequence of mappings from SQL <identifier> to XML Name 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>.

5 Lexical elements

This Clause modifies Clause 5, "Lexical elements", in ISO/IEC 9075-2.

5.1 <token> and <separator>

This Subclause modifies Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2.

Function

Specify lexical units (tokens and separators) that participate in SQL language.

Format

```
<non-reserved word> ::=
         !! All alternatives from ISO/IEC 9075-2

<reserved word> ::=
         !! All alternatives from ISO/IEC 9075-2
       | XML | XMLAGG | XMLATTRIBUTES | XMLCONCAT | XMLELEMENT | XMLFOREST | XMLGEN
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

6 Scalar expressions

This Clause modifies Clause 6, "Scalar expressions", in ISO/IEC 9075-2.

6.1 <data type>

This Subclause modifies Subclause 6.1, "<data type>", in ISO/IEC 9075-2.

Function

Specify a data type.

Format

```
<predefined type> ::=

  !! All alternatives from ISO/IEC 9075-2

| <XML type>

<XML type> ::= XML
```

Syntax Rules

1) Insert after SR 39) XML specifies the data type XML.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 21) If <data type> is an <XML type>, then an XML data type descriptor is created, including the name of the data type (XML).

Conformance Rules

1) Insert this CR Without Feature X001, “Elementary XML type”, a <predefined type> shall not specify XML.

6.2 <cast specification>

This Subclause modifies Subclause 6.12, "<cast specification>", in ISO/IEC 9075-2.

Function

Specify a data conversion.

Format

No additional Format items.

Syntax Rules

1) Replace 6) If the <cast operand> is a <value expression>, then the valid combinations of *TD* and *SD* in a <cast specification> are given by the following table. "Y" indicates that the combination is syntactically valid without restriction; "M" indicates that the combination is valid subject to other Syntax Rules in this Subclause being satisfied; and "N" indicates that the combination is not valid:

<data type> *SD* of <value expression>	<data type> of *TD*																
	EN	AN	VC	FC	D	T	TS	YM	DT	BO	UDT	CL	BL	RT	CT	RW	XML
EN	Y	Y	Y	Y	N	N	N	M	M	N	M	Y	N	M	N	N	N
AN	Y	Y	Y	Y	N	N	N	N	N	N	M	Y	N	M	N	N	N
C	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	M	Y	N	M	N	N	N
D	N	N	Y	Y	Y	N	Y	N	N	N	M	Y	N	M	N	N	N
T	N	N	Y	Y	N	Y	Y	N	N	N	M	Y	N	M	N	N	N
TS	N	N	Y	Y	Y	Y	Y	N	N	N	M	Y	N	M	N	N	N
YM	M	N	Y	Y	N	N	N	Y	N	N	M	Y	N	M	N	N	N
DT	M	N	Y	Y	N	N	N	N	Y	N	M	Y	N	M	N	N	N
BO	N	N	Y	Y	N	N	N	N	N	Y	M	Y	N	M	N	N	N
UDT	M	M	M	M	M	M	M	M	M	M	M	M	M	M	N	N	N
BL	N	N	N	N	N	N	N	N	N	N	M	N	Y	M	N	N	N
RT	M	M	M	M	M	M	M	M	M	M	M	M	M	M	N	N	N
CT	N	N	N	N	N	N	N	N	N	N	N	N	N	N	M	N	N
RW	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	M	N
XML	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y

Where:

EN = Exact Numeric
AN = Approximate Numeric
C = Character (Fixed- or Variable-length, or character large object)
FC = Fixed-length Character
VC = Variable-length Character
CL = Character Large Object
D = Date
T = Time
TS = Timestamp
YM = Year-Month Interval
DT = Day-Time Interval
BO = Boolean
UDT = User-Defined Type
BL = Binary Large Object
RT = Reference type
CT = Collection type
RW = Row type
XML = XML

NOTE 3 – If *SD* is XML and *TD* is not XML, or if *SD* is not XML and *TD* is XML, then the combination of *SD* and *TD* is not valid unless it is explicitly defined to be valid in another part of ISO/IEC 9075.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR If *SD* is XML, then *TV* is *SV*.

WG3:DRS-020 = H2-2002-365
6.2 <cast specification>

Conformance Rules

1) Insert this CR Without Feature X001, "Elementary XML type", the declared type of <cast operand> shall not be XML.

2) Insert this CR Without Feature X001, "Elementary XML type", <cast target> shall not be XML.

6.3 <value expression>

This Subclause modifies Subclause 6.25, "<value expression>", in ISO/IEC 9075-2.

Function

Specify a value.

Format

```
<common value expression> ::=
         !! All alternatives from ISO/IEC 9075-2
       | <XML value expression>
```

Syntax Rules

1) Replace SR 2) The declared type of a <common value expression> is the declared type of the <numeric value expression>, <string value expression>, <datetime value expression>, <interval value expression>, <user-defined type value expression>, <collection value expression>, <reference value expression>, or <XML value expression>, respectively.

2) Insert after SR 8)n) An <aggregate function> that specifies <XML aggregate>.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 2) The value of a <common value expression> is the value of the immediately contained <numeric value expression>, <string value expression>, <datetime value expression>, <interval value expression>, <user-defined type value expression>, <collection value expression>, or <reference value expression>, or <XML value expression>.

Conformance Rules

1) Insert this CR Without Feature X001, “Elementary XML type”, a <common value expression> shall not be an <XML value expression>.

6.4 <XML value expression>

Function

Specify an XML value.

Format

```
<XML value expression> ::= <XML primary>

<XML primary> ::=
         <value expression primary>
       | <XML value function>
```

Syntax Rules

1) The declared type of <value expression primary> shall be XML.

2) The declared type of <XML value expression> is XML.

Access Rules

None.

General Rules

1) The value of <XML value expression> is the value of the simply contained <value expression primary> or <XML value function>.

Conformance Rules

1) Without Feature X001, “Elementary XML type”, an <XML value expression> shall not be specified.

6.5 <XML value function>

Function

Specify a function that yields a value of type XML.

Format

```
<XML value function> ::=
         <XML concatenation>
       | <XML element>
       | <XML forest>
       | <XML gen>
```

Syntax Rules

1) The declared type of <XML value function> is XML.

Access Rules

None.

General Rules

1) The result of an <XML value function> is the XML value of the immediately contained <XML forest>, <XML concatenation>, <XML element>, or <XML gen>.

Conformance Rules

1) Without Feature X001, “Elementary XML type”, an <XML value function> shall not be specified.

6.6 <XML concatenation>

Function

Concatenate a list of XML values.

Format

```
<XML concatenation> ::=
       XMLCONCAT <left paren> <XML value expression>
         { <comma> <XML value expression> }... <right paren>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let n be the number of <XML value expression>s immediately contained in <XML concatenation>. Let XV_1, XV_2, . . . , XV_n be the results of these <XML value expression>s.

2) If any of XV_1, XV_2, . . . , XV_n is an XML document, and any other value in XV_1, XV_2, . . . , XV_n is non-null, then an exception condition is raised: *data exception — invalid XML value in XML operation*.

3) Let R be an empty forest of elements.

4) If, for all i, 1 (one) $\leq i \leq n$, XV_i is the null value, then let R be the null value; otherwise, let R be the result of the concatenation of R and XV_i, 1 (one) $\leq i \leq n$.

   NOTE 4 – "concatenation" of XML values is defined in Subclause 4.2.1, "Characteristics of XML values", Table 1, "Concatenation of two XML values".

5) The result of <XML concatenation> is R.

Conformance Rules

1) Without Feature X001, "Elementary XML type", an <XML concatenation> shall not be specified.

6.7 <XML element>

Function

Generate an XML element.

Format

```
<XML element> ::=
       XMLELEMENT <left paren>
         NAME <XML element name>
           [ <comma> <XML attributes> ]
           [ { <comma> <XML element content> }... ]
       <right paren>

<XML element name> ::= <identifier>

<XML attributes> ::=
       XMLATTRIBUTES <left paren> <XML attribute list> <right paren>

<XML attribute list> ::=
       <XML attribute> [ { <comma> <XML attribute> }... ]

<XML attribute> ::=
       <XML attribute value> [ AS <XML attribute name> ]

<XML attribute value> ::= <value expression>

<XML attribute name> ::= <identifier>

<XML element content> ::= <value expression>
```

Syntax Rules

1) If an <XML attribute name> is not specified in an <XML attribute>, then the <value expression> simply contained in that <XML attribute> shall be a <column reference>.

Access Rules

None.

General Rules

1) Let n be the number of occurrences of <XML attribute> in <XML attribute list> that have an <XML attribute value> that is not the null value.

2) Let i range from 1 (one) to n.

   a) Let A_i be the i-th non-null <XML attribute> contained in <XML attribute list>.

   b) Let AV_i be the <XML attribute value> of A_i. If A_i contains an <XML attribute name>, then let AN_i be that <XML attribute name>; otherwise, let AN_i be the zero-length string.

   c) Let ANC_i be the character representation of the name of A_i.

Case:

i) If AC_i is not the zero-length string, then ANC_i is the result of the partially escaped mapping from an SQL <identifier> to an XML Name specified in Subclause 7.1, "Mapping SQL <identifier>s to XML names", applied to AN_i.

ii) Otherwise, let CN_i be the <column name> of the column designated by the <column reference> that is A_i. ANC_i is the result of the fully escaped mapping from an SQL <identifier> to an XML Name specified in Subclause 7.1, "Mapping SQL <identifier>s to XML names", applied to CN_i.

3) Let m be the number of occurrences of <XML element content> in <XML element> that do not have the null value. Let EC_j, 1 (one) $\leq j \leq m$, be the non-null <XML element content>s contained in <XML element>.

4) Let *EN* be the character representation of <XML element name> after the partially escaped mapping from an SQL <identifier> to an XML Name specified in Subclause 7.1, "Mapping SQL <identifier>s to XML names", has been applied.

5) Let *XGT* be the <character string literal> that has the value of the following <string value expression>:

```
'<' || 'EN'
|| ' ANC1="{$AVAR1}"' || ... || ' ANCn="{$AVARn}"' || '>'
|| '{$ECVAR1}' || ... || '{$ECVARm}'
|| '</' || 'EN' || '>'
```

6) The result of <XML element> is the result of the following <XML gen>:

```
XMLGEN ( XGT,
         AV1 AS "AVAR1", ..., AVn AS "AVARn",
         EC1 AS "ECVAR1", ..., ECm AS "ECVARm"
       )
```

Conformance Rules

1) Without Feature X001, "Elementary XML type", an <XML element> shall not be specified.

6.8 <XML forest>

Function

Generate an XML forest of elements.

Format

```
<XML forest> ::=
       XMLFOREST <left paren> <forest element list> <right paren>

<forest element list> ::=
       <forest element> [ { <comma> <forest element> }... ]

<forest element> ::=
       <forest element value> [ AS <forest element name> ]

<forest element value> ::= <value expression>

<forest element name> ::= <identifier>
```

Syntax Rules

1) If a <forest element name> is not specified in a <forest element>, then the <forest element value> shall be a <column reference>.

2) Let n be the number of occurrences of <forest element>s in <forest element list>. For each i between 1 (one) and n:

   a) Let F_i be the i-th <forest element>s contained in <forest element list>.

   b) Let FV_i be the <forest element value> of F_i.

   c) Case:

      i) If F_i contains a <forest element name> FEN_i, then let FNC_i be the result of the partially escaped mapping from an SQL <identifier> to an XML Name specified in Subclause 7.1, "Mapping SQL <identifier>s to XML names".

      ii) Otherwise, let CN_i be the <column name> of the column designated by the <column reference> that is F_i. Let FNC_i be the result of the fully escaped mapping from an SQL <identifier> to an XML Name specified in Subclause 7.1, "Mapping SQL <identifier>s to XML names", applied to CN_i.

   d) Let XCA_i be:

```
CASE WHEN FV_i IS NULL THEN NULL
     ELSE XMLGEN ('<FNC_i>{$FV}</FNC_i >', FV_i AS FV) END
```

3) <XML forest> is equivalent to the following <XML concatenation>:

```
XMLCONCAT ( XCA_1,
            ... ,
            XCA_n )
```

6.8 <XML forest>

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature X001, “Elementary XML type”, an <XML forest> shall not be specified.

6.9 <XML gen>

Function

<XML gen> constructs an XML element using an XQuery constructor.

Format

```
<XML gen> ::=
      XMLGEN <left paren>
        <XML constructor>
        [ <XML gen variable list> ]
      <right paren>

<XML constructor> ::= <string value expression>

<XML gen variable list> ::=
      { <comma> <XML gen variable> }...

<XML gen variable> ::=
      <XML gen variable value> [ AS <XML gen variable name> ]

<XML gen variable value> ::= <value expression>

<XML gen variable name> ::= <identifier>
```

Syntax Rules

1) If an <XML gen variable name> is not specified in an <XML gen variable>, then the <value expression> simply contained in that <XML gen variable> shall be a <column reference>.

2) Let *n* be the number of occurrences of <XML gen variable> in <XML gen variable list>.

3) Let *i* range from 1 (one) to *n*.

   a) Let V_i be the *i*-th <XML gen variable> contained in <XML gen variable list>.

   b) Let VV_i be the <XML gen variable value> of V_i. If VV_i contains an <XML gen variable name>, then let VN_i be that <XML gen variable name>; otherwise, let VN_i be the zero-length string.

   c) Let VNC_i be

      Case:

      i) If <XML gen variable name> is specified, then the result of the partially escaped mapping from an SQL <identifier> to an XML Name specified in Subclause 7.1, “Mapping SQL <identifier>s to XML names”, has been applied.

      ii) Otherwise, the result of the fully escaped mapping from an SQL <identifier> to an XML Name specified in Subclause 7.1, “Mapping SQL <identifier>s to XML names”) , applied to the <column name> of the column designated by the <column reference> that is V_i.

   d) If *i* is greater than 1 (one), then let *j* range from 1 (one) to $i-1$. VNC_i shall not be equal to any VNC_j according to the rules of Subclause 8.2, "<comparison predicate>", in ISO/IEC 9075-2.

e) The format of VNC_i shall satisfy rule 189, "QName", specified in *XQuery 1.0: An XML Query Language*.

Access Rules

None.

General Rules

1) Let *XMLC* be the value of the <XML constructor>.

2) If *XMLC* does not satisfy the format of <XQuery constructor>, then an exception condition is raised: *data exception — invalid XQuery constructor*.

3) If any <XQuery ExprSequence> contained in an <XQuery EnclosedExpr> contained in *XMLC* is not an <XQuery Variable>, then an exception condition is raised: *data exception — invalid XQuery constructor*.

   NOTE 5 – An <XQuery EnclosedExpr> is an XQuery expression within curly braces; this rule restricts all { } items in the <XQuery Constructor> to reference <XQuery Variable>s available in the XQuery evaluation context of the <XQuery Constructor>.

4) For i between 1 (one) and n, let XV_i be

   Case:

   a) If the value of VV_i is null, then the XQuery empty sequence.

   b) Otherwise, the result of applying the mapping defined in Subclause 7.16, "Mapping SQL data values to XML", to the value of VV_i.

5) For i between 1 (one) and n, XV_i is bound to an <XQuery Variable> named "$\$VNC_i$" in the XQuery evaluation context of *XMLT*.

   NOTE 6 – XQuery defines how a value is bound to an <XQuery Variable>.

6) The result *XMLR* of <XML gen> is the XML value that is determined by applying General Rules of Subclause 9.2, "XQuery syntactic elements", to *XMLC* using the values bound to the <XQuery Variable>s named "$\$VNC_i$" for i between 1 (one) and n. If the result is an XQuery error, then an exception condition is raised: *data exception — XMLGEN evaluation failure*.

7) If *XMLR* is not well-formed, then an exception condition is raised: *data exception — XML value not well-formed*.

   NOTE 7 – This rule assures that no attribute name within an element is used more than once.

Conformance Rules

1) Without Feature X001, "Elementary XML type", an <XML gen> shall not be specified.

2) Without Feature X002, "Advanced XML type", <XML constructor> shall be a <literal>.

7 Mappings

7.1 Mapping SQL <identifier>s to XML names

Function

Define the mapping of SQL <identifier>s to XML Names.

Format

```
<uppercase hexit> ::= <digit> | A | B | C | D | E | F
```

Syntax Rules

None.

General Rules

1) Let *SQLI* be an SQL <identifier> in an application of this Subclause. *SQLI* is a sequence of characters of SQL_TEXT. Let N be the number of characters in *SQLI*. Let S_1, S_2, . . . , S_N be the characters of *SQLI*, in order from left to right.

2) Let *EV* be the escape variant in an application of this Subclause. *EV* is either *partially escaped* or *fully escaped*.

3) Let *TM* be the implementation-defined mapping of the characters of SQL_TEXT to characters of Unicode.

   NOTE 8 – Unicode scalar values in the ranges U+0000 through U+001F (inclusive), sometimes called the "C0 controls", and U+007F through U+009F (inclusive), sometimes called "delete" (U+007F) and the "C1 controls" (the remainder of that latter range) are not encoding of abstract characters in Unicode. Programs that conform to the Unicode Standard may treat these Unicode scalar values in exactly the same way as they treat the 7- and 8-bit equivalents in other protocols. Such usage constitutes a higher-level protocol and is beyond the scope of the Unicode standard. These Unicode scalar values do not occur in XML Names, but may appear in other places in XML text.

4) For each i between 1 (one) and N, let $\mathbf{T}_i$ be $TM(S_i)$.

5) For each i between 1 (one) and N, let $\mathbf{X}_i$ be the Unicode character string defined by the following rules.

   Case:

   a) If S_i has no mapping to Unicode (*i.e.*, $TM(S_i)$ is undefined), then $\mathbf{X}_i$ is implementation-defined.

   b) If S_i is <colon>, then

      Case:

      i) If i = 1 (one), then let $\mathbf{X}_i$ be **_x003A_**.

ii) If *EV* is fully escaped, then let ***X***$_i$ be _**x003A**_.

iii) Otherwise, let ***X***$_i$ be ***T***$_i$.

c) If $i \leq N-1$, S_i is <underscore>, and S_{i+1} is the lowercase letter **x**, then let ***X***$_i$ be _**x005F**_.

d) If *EV* is fully escaped, $i = 1$ (one), $N \geq 3$, S_1 is either the uppercase letter **X** or the lowercase letter **x**, S_2 is either the uppercase letter **M** or the lowercase letter **m**, and S_3 is either the uppercase letter **L** or the lowercase letter **l**, then

Case:

a) If S_1 is the lowercase letter **x**, then let ***X***$_1$ be _**x0078**_.

b) If S_1 is the uppercase letter **X**, then let ***X***$_1$ be _**x0058**_.

e) If ***T***$_i$ is not a valid XML NameChar, or if $i = 1$ (one) and ***T***$_1$ is not a valid first character of an XML Name, then:

i) Let ***U***$_1$, ***U***$_2$, . . . , ***U***$_8$ be the eight <uppercase hexit>s such that ***T***$_i$ is U+***U***$_1 U_2 ... U_8$ in the UCS-4 encoding.

ii) Case:

1) If ***U***$_1$ = **0**, ***U***$_2$ = **0**, ***U***$_3$ = **0**, and ***U***$_4$ = **0**, then let **X**$_i$ be _**x*U*$_5$*U*$_6$*U*$_7$*U*$_8$**_.
NOTE 9 – This case implies that ***T***$_i$ has a UCS-2 encoding, which is U+***U***$_5 U_6 U_7 U_8$.

2) Otherwise, let ***X***$_i$ be _**x*U*$_1$*U*$_2$*U*$_3$*U*$_4$*U*$_5$*U*$_6$*U*$_7$*U*$_8$**_.
NOTE 10 – The normative definition of valid XML Name characters is found in Extensible Markup Language (XML) Version 1.0 (second edition), as cited in Subclause 2.2, "Publicly-available specifications". Valid first characters of XML Names are Letters, <underscore> and <colon>. Valid XML Name characters, after the first character, are Letters, Digits, <period>, <minus sign>, <underscore>, <colon>, CombiningChars, and Extenders. Note that the XML definition of Letter and Digit is broader than <simple Latin letter> and <digit> respectively.

f) Otherwise, let ***X***$_i$ be ***T***$_i$.
NOTE 11 – That is, any character in *SQLI* that does not occasion a problem as a character in an XML Name is simply copied into the XML Name.

6) Let ***XMLN*** be the character string concatenation of ***X***$_1$, ***X***$_2$, . . . , and ***X***$_N$ in order from left to right.

7) ***XMLN*** is the XML Name that is the mapping of *SQLI* to XML.

Conformance Rules

None.

7.2 Mapping a multi-part SQL name to an XML name

Function

Define the mapping of a sequence of SQL <identifier>s to an XML Name.

General Rules

1) Let $SQLI_i$, 1 (one) $\leq i \leq n$ be a sequence of n SQL <identifier>s provided for an application of this Subclause.

2) Let NP(S) be the mapping of a string S to a result string defined as follows:

   a) Let m be the number of characters in S. For each character S_j, 1 (one) $\leq j \leq m$, in S, let NPS_j be defined as follows:

      i) If S_j is <period>, then NPS_j is "_x002E_".

      ii) Otherwise, NPS_j is S_j.

   b) NP(S) is the concatenation of NPS_j, 1 (one) $\leq j \leq m$.

3) For each i between 1 (one) and n, let $XMLN_i$ be the XML Name formed by the application of Subclause 7.1, "Mapping SQL <identifier>s to XML names", to $SQLI_i$ using the fully escaped variant of the mapping.

4) Let ***XMLR*** be the concatenation of the following strings:

   NP(*XMLN*$_1$), <period>, NP(*XMLN*$_2$), <period>, ... NP(*XMLN*$_n$)

5) ***XMLR*** is the XML Name that is the result of this mapping.

Conformance Rules

None.

7.3 Mapping an SQL table to an XML document and an XML schema document

Function

Define the mapping of an SQL table to an XML document and an XML Schema document that describes this XML document.

General Rules

1) Let *T* be the table provided for an application of this mapping. 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). Let *U* be the authorization identifier that is invoking this mapping.

2) Let ***XS*** be the XML Schema document that is the result of this mapping. ***XS*** reflects the metadata associated with *T*.

   a) Let *TC*, *TS*, and *TN* be the <catalog name>, <unqualified schema name>, and <qualified identifier> of the <table name> of *T*, respectively.

   b) Let ***XMLTN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *TN* using the fully escaped variant of the mapping.

   c) Let ***XMLTYPEN*** be the result of applying the mapping defined in Subclause 7.2, "Mapping a multi-part SQL name to an XML name", to "TableType", *TC*, *TS*, and *TN*.

   d) Let *CT* be the visible columns of *T* for *U*. Let ***XSCT*** be the result of applying the mapping defined in Subclause 7.10, "Mapping a collection of SQL data types to XML schema data types", to the data types and domains of the columns of *CT* using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with **xsi:nil='true'**.

   e) Let ***XST*** be the result of applying the mapping defined in Subclause 7.6, "Mapping an SQL table to XML schema data types", to *T* using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with **xsi:nil="true"** and *U* as the invoker of this mapping.

   f) Let ***SQLXMLNS*** be the value of the namespace definition provided for the namespace variable **sqlxml:** in Table 2, "Namespace prefixes and their URIs".

   g) Let ***XSDNS*** be the value of the namespace definition provided for the namespace variable **xsd:** in Table 2, "Namespace prefixes and their URIs".

   h) ***XS*** has the following contents:

```
<?xml version="1.0"?>

<xsd:schema
      xmlns:xsd="XSDNS"
      xmlns:sqlxml="SQLXMLNS">

   <xsd:import
      namespace="SQLXMLNS"
      schemaLocation="SQLXMLNS.xsd" />

   XSCT
```

```
   XST
   <xsd:element name="XMLTN" type="XMLTYPEN" />
</xsd:schema>
```

Editor's Note
Document *XS* is created without declaring a namespace. A user may wish to specify a namespace for this mapping that is used as the value of an **xsd:targetNamespace** attribute. See Possible Problem **XML-007** in the Editor's Notes.

3) Let ***XSL*** be the URL that identifies ***XS***.

4) Let ***XD*** be the XML document that is the result of this mapping. ***XD*** reflects the data of *T*.

   a) Let ***XDROWS*** be the result of applying the mapping defined in Subclause 7.12, "Mapping an SQL table to an XML element", to *T* using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with **xsi:nil="true"** and *U* as the invoker of this mapping.

   b) Let ***XSINS*** be the value of the namespace definition provided for the namespace variable **xsi:** in Table 2, "Namespace prefixes and their URIs".

   c) ***XD*** has the following contents:

```
<?xml version="1.0"?>
<XMLTN
   xmlns:xsi="XSINS"
   xsi:noNamespaceSchemaLocation="XSL">
   XDROWS
</XMLTN>
```

5) ***XD*** is the XML document and ***XS*** is the XML Schema document that describes ***XD*** that are the result of this mapping.

Conformance Rules

None.

7.4 Mapping an SQL schema to an XML document and an XML schema document

Function

Define the mapping of an SQL schema to an XML document and an XML Schema document that describes this XML document.

General Rules

1) Let *S* be the schema provided for an application of this mapping. 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). Let *U* be the authorization identifier that is invoking this mapping.

2) Let ***XS*** be the XML Schema document that is the result of this mapping. ***XS*** reflects the metadata associated with *S*.

   a) Let *SC* and *SN* be the <catalog name> and <unqualified schema name> of *S*, respectively.

   b) Let ***XMLSN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *SN* using the fully escaped variant of the mapping.

   c) Let *CT* be the visible columns of the viewed and base tables contained in *S* for *U*. Let ***XSCT*** be the result of applying the mapping defined in Subclause 7.10, "Mapping a collection of SQL data types to XML schema data types", to the data types and domains of the columns of *CT* using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with **xsi:nil='true'**.

   d) Let *XMLTYPEN* be the result of applying the mapping defined in Subclause 7.2, "Mapping a multi-part SQL name to an XML name", to "SchemaType", *SC*, and *SN*.

   e) Let ***XST*** be the result of applying the mapping defined in Subclause 7.7, "Mapping an SQL schema to XML schema data types", to *S* using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with **xsi:nil="true"** and *U* as the invoker of this mapping.

   f) Let ***SQLXMLNS*** be the value of the namespace definition provided for the namespace variable **sqlxml:** in Table 2, "Namespace prefixes and their URIs".

   g) Let ***XSDNS*** be the value of the namespace definition provided for the namespace variable **xsd:** in Table 2, "Namespace prefixes and their URIs".

   h) ***XS*** has the following contents:

```
<?xml version="1.0"?>

<xsd:schema
      xmlns:xsd="XSDNS"
      xmlns:sqlxml="SQLXMLNS">

   <xsd:import
      namespace="SQLXMLNS"
      schemaLocation="SQLXMLNS.xsd" />

   XSCT

   XST
```

```
    <xsd:element name="XMLSN" type="XMLTYPEN" />
</xsd:schema>
```

3) Let ***XSL*** be the URL that identifies ***XS***.

4) Let ***XD*** be the XML Document that is the result of this mapping. ***XD*** reflects the data of the tables contained in *S*.

   a) Let ***XDSCHEMA*** be the result of applying the mapping defined in Subclause 7.13, "Mapping an SQL schema to an XML element", to *S* using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with **xsi:nil="true"** and *U* as the invoker of this mapping.

   b) Let ***XSINS*** be the value of the namespace definition provided for the namespace variable **xsi:** in Table 2, "Namespace prefixes and their URIs".

   c) ***XD*** has the following contents:

```
<?xml version="1.0"?>

<XMLSN
   xmlns:xsi="XSINS"
   xsi:noNamespaceSchemaLocation="XSL">

   XDSCHEMA

</XMLSN>
```

5) ***XD*** is the XML document and ***XS*** is the XML Schema document that describes ***XD*** that are the result of this mapping.

Conformance Rules

None.

7.5 Mapping an SQL catalog to an XML document and an XML schema document

Function

Define the mapping of an SQL catalog to an XML document and an XML Schema document that describes this XML document.

General Rules

1) Let *C* be the catalog provided for an application of this mapping. 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). Let *U* be the authorization identifier that is invoking this mapping.

2) Let ***XS*** be the XML Schema document that is the result of this mapping. ***XS*** reflects the metadata associated with *C*.

   a) Let *CN* be the <catalog name> of *C*.

   b) Let ***XMLCN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *CN* using the fully escaped variant of the mapping.

   c) Let *CT* be the visible columns of the viewed and base tables contained in C for *U*. Let ***XSCT*** be the result of applying the mapping defined in Subclause 7.10, "Mapping a collection of SQL data types to XML schema data types", to the data types and domains of the columns of *CT* using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with **xsi:nil='true'**.

   d) Let ***XMLTYPEN*** be the result of applying the mapping defined in Subclause 7.2, "Mapping a multi-part SQL name to an XML name", to "CatalogType" and *CN*.

   e) Let ***XST*** be the result of applying the mapping defined in Subclause 7.8, "Mapping an SQL catalog to XML schema data types", to *C* using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with **xsi:nil="true"** and *U* as the invoker of this mapping.

   f) Let ***SQLXMLNS*** be the value of the namespace definition provided for the namespace variable **sqlxml:** in Table 2, "Namespace prefixes and their URIs".

   g) Let ***XSDNS*** be the value of the namespace definition provided for the namespace variable **xsd:** in Table 2, "Namespace prefixes and their URIs".

   h) ***XS*** has the following contents:

```
<?xml version="1.0"?>

<xsd:schema
      xmlns:xsd="XSDNS"
      xmlns:sqlxml="SQLXMLNS">

   <xsd:import
      namespace="SQLXMLNS"
      schemaLocation="SQLXMLNS.xsd" />

   XSCT

   XST
```

```
    <xsd:element name="XMLCN" type="XMLTYPEN" />
</xsd:schema>
```

3) Let ***XSL*** be the URL that identifies ***XS***.

4) Let ***XD*** be the XML Document that is the result of this mapping. *XD* reflects the data of the tables contained in *C*.

   a) Let ***XDCATALOG*** be the result of applying the mapping defined in Subclause 7.14, "Mapping an SQL catalog to an XML element", to *C* using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with **xsi:nil="true"** and *U* as the invoker of this mapping.

   b) Let ***XSINS*** be the value of the namespace definition provided for the namespace variable **xsi:** in Table 2, "Namespace prefixes and their URIs".

   c) ***XD*** has the following contents:

```
<?xml version="1.0"?>
<XMLCN
   xmlns:xsi="XSINS"
   xsi:noNamespaceSchemaLocation="XSL">
   XDCATALOG
</XMLCN>
```

5) ***XD*** is the XML document and ***XS*** is the XML Schema document that describes ***XD*** that are the result of this mapping.

Conformance Rules

None.

7.6 Mapping an SQL table to XML schema data types

Function

Define the mapping of an SQL table to XML Schema data types.

General Rules

1) Let *T* be the table provided for an application of this mapping. 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). Let *U* be the authorization identifier that is invoking this mapping.

2) Let *TC*, *TS*, and *TN* be the <catalog name>, <unqualified schema name>, and <qualified identifier> of the <table name> of *T*, respectively.

3) Let ***XMLTN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *TN* using the fully escaped variant of the mapping.

4) Let *n* be the number of visible columns of *T* for *U*.

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

   a) Let C_i be the *i*-th visible column of *T* for *U* in order of its ordinal position within *T*.

   b) Let *CN* be the <column name> of C_i. Let *D* be the data type of C_i.

   c) Let ***XMLCN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *CN* using the fully escaped variant of the mapping.

   d) Let ***XMLCTN*** be the result of applying the mapping defined in Subclause 7.9, "Mapping an SQL data type to an XML name", to *D*.

   e) Case:

      i) If C_i is known not nullable, then let ***XMLNULLS*** be the zero-length string.

      ii) Otherwise,

      Case:

      1) If *NULLS* is absent, then let ***XMLNULLS*** be

      `minOccurs="0"`

      2) If *NULLS* is nil, then let ***XMLNULLS*** be

      `nillable="true"`

   f) Case:

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

      1) Let *CS* be the character set of *D*.

      2) Let *CSC*, *CSS*, and *CSN* be the <catalog name>, <unqualified schema name>, and <SQL language identifier> of the <character set name> of *CS*, respectively.

3) Let ***XMLCSCN***, ***XMLCSSN***, and ***XMLCSN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *CSC*, *CSS*, and *CSN*, respectively, using the fully escaped variant of the mapping.

4) Let *CO* be the collation of *D*.

5) Let *COC*, *COS*, and *CON* be the <catalog name>, <unqualified schema name>, and <qualified identifier> of the <collation name> of *CO*, respectively.

6) Let ***XMLCOCN***, ***XMLCOSN***, and ***XMLCON*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *COC*, *COS*, and *CON*, respectively, using the fully escaped variant of the mapping.

7) It is implementation-dependant whether *COLANN* is the zero-length string or

```
<xsd:annotation>
   <xsd:appinfo>
      <sqlxml:sqlname
         type="CHARACTER SET"
         catalogName="XMLCSCN"
         schemaName="XMLCSSN"
         localName="XMLCSN" />
      <sqlxml:sqlname
         type="COLLATION"
         catalogName="XMLCOCN"
         schemaName="XMLCOSN"
         localName="XMLCON" />
   </xsd:appinfo>
</xsd:annotation>
```

ii) Otherwise, let ***COLANN*** be the zero-length string.

g) Let ***XMLCE***$_i$ be

```
<xsd:element name="XMLCN" type="XMLCTN" XMLNULLS>
   COLANN
</xsd:element>
```

6) Let ***XMLTYPEN*** be the result of applying the mapping defined in Subclause 7.2, "Mapping a multi-part SQL name to an XML name", to "TableType", *TC*, *TS*, and *TN*.

7) Let ***XMLROWN*** be the result of applying the mapping defined in Subclause 7.2, "Mapping a multi-part SQL name to an XML name", to "RowType", *TC*, *TS*, and *TN*.

8) Let ***XMLCN***, ***XMLSN***, and ***XMLTN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *TC*, *TS*, and *TN*, respectively, using the fully escaped variant of the mapping.

9) If *T* is a base table, then ***TYPE*** is **BASE TABLE**. Otherwise, ***TYPE*** is **VIEWED TABLE**. It is implementation-dependent whether ***SQLANN*** is the zero-length string or

```
<xsd:annotation>
   <xsd:appinfo>
      <sqlxml:sqlname
          type="TYPE"
          catalogName="XMLCN"
          schemaName="XMLSN"
          localName="XMLTN" />
   </xsd:appinfo>
</xsd:annotation>
```

10) Let ***XMLTN*** be:

```
<xsd:complexType name="XMLROWN">
   <xsd:sequence>
      XMLCE1
      ...
      XMLCEn
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="XMLTYPEN">
   SQLANN
   <xsd:sequence>
      <xsd:element name="row"
         type="XMLROWN"
         minOccurs="0"
         maxOccurs="unbounded" />
   </xsd:sequence>
</xsd:complexType>
```

Editor's Note
Some participants feel that the row element is not necessary. Instead of multiple row elements within a table element, they would create multiple elements with the name of the table. See Possible Problem **XML-008** in the Editor's Notes.

11) ***XMLTN*** contains the XML Schema data types that are the result of this mapping.

Conformance Rules

None.

7.7 Mapping an SQL schema to XML schema data types

Function

Define the mapping of an SQL table to XML Schema data types.

General Rules

1) Let *S* be the schema provided for an application of this mapping. 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). Let *U* be the authorization identifier that is invoking this mapping.

2) Let *SC*, and *SN* be the <catalog name> and <unqualified schema name> of the <schema name> of *S*, respectively.

3) Let ***XMLSN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *SN* using the fully escaped variant of the mapping.

4) Let *n* be the number of visible tables of *S* for *U*.

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

   a) Let T_i be the *i*-th visible table of *S*.

   b) Let *TN* be the <qualified identifier> of the <table name> of T_i.

   c) Let ***XMLTN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *TN* using the fully escaped variant of the mapping.

   d) Let ***XMLTTN*** be the result of applying the mapping defined in Subclause 7.2, "Mapping a multi-part SQL name to an XML name", to "TableType", *SC*, *SN*, and *TN*.

   e) Let $\textbf{\textit{XMLTE}}_i$ be

```
<xsd:element name="XMLTN" type="XMLTTN" />
```

6) Let ***XMLTYPEN*** be the result of applying the mapping defined in Subclause 7.2, "Mapping a multi-part SQL name to an XML name", to "SchemaType", *SC*, and *SN*.

7) Let ***XMLSC*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *SC* using the fully escaped variant of the mapping.

8) It is implementation-dependant whether ***SQLANN*** is the zero-length string or

```
<xsd:annotation>
   <xsd:appinfo>
      <sqlxml:sqlname
          type="SCHEMA"
          catalogName="XMLSC"
          schemaName="XMLSN" />
   </xsd:appinfo>
</xsd:annotation>
```

9) Let ***XMLSCHEMAT*** be:

```
<xsd:complexType name="XMLTYPEN">
   SQLANN
   <xsd:all>
      XMLTE1
      ...
      XMLTEn
   </xsd:all>
</xsd:complexType>
```

10) ***XMLSCHEMAT*** contains the XML Schema data types that are the result of this mapping.

Conformance Rules

None.

7.8 Mapping an SQL catalog to XML schema data types

Function

Define the mapping of an SQL catalog to XML Schema data types.

General Rules

1) Let *C* be the catalog provided for an application of this mapping. 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). Let *U* be the authorization identifier that is invoking this mapping.

2) Let *CN* be the <catalog name> of *C*.

3) Let ***XMLCN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *CN* using the fully escaped variant of the mapping.

4) Let *n* be the number of visible schemas of *C* for *U*.

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

   a) Let S_i be the *i*-th visible schema of *S*.

   b) Let *SN* be the <unqualified schema name> of the <schema name> of S_i.

   c) Let ***XMLSN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *SN* using the fully escaped variant of the mapping.

   d) Let ***XMLSTN*** be the result of applying the mapping defined in Subclause 7.2, "Mapping a multi-part SQL name to an XML name", to "SchemaType", *SC*, and *SN*.

   e) Let $\textbf{\textit{XMLSE}}_i$ be

```
<xsd:element name="XMLSN" type="XMLSTN" />
```

6) Let ***XMLTYPEN*** be the result of applying the mapping defined in Subclause 7.2, "Mapping a multi-part SQL name to an XML name", to "CatalogType" and *CN*.

7) It is implementation-dependant whether ***SQLANN*** is the zero-length string or

```
<xsd:annotation>
   <xsd:appinfo>
      <sqlxml:sqlname
          type="CATALOG"
          catalogName="XMLCN" />
   </xsd:appinfo>
</xsd:annotation>
```

8) Let ***XMLCATT*** be:

```
<xsd:complexType name="XMLTYPEN">
   SQLANN
   <xsd:all>
      XMLSE1
      ...
      XMLSEn
   </xsd:all>
</xsd:complexType>
```

9) ***XMLCATT*** contains the XML Schema data types that are the result of this mapping.

Conformance Rules

None.

7.9 Mapping an SQL data type to an XML name

Function

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

General Rules

1) Let *D* be the SQL data type or 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 Unicode of *CSM*(*S*), for all strings *S* of length *N* of characters of *SQLCS*.

   d) Let ***MLIT*** be the canonical XML Schema literal denoting *MAXCSL* in the lexical representation of XML Schema type **xsd:integer**.

   e) Let ***NLIT*** be the canonical XML Schema literal denoting *N* in the lexical representation of XML Schema type **xsd: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 **xsd: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 **xsd: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 **xsd: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 **xsd: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 **xsd: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 **xsd: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 **xsd: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 **xsd: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 ***XTPS*** be the canonical XML Schema literal denoting TPS in the lexical representation of XML Schema type **xsd: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 **xsd: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) the type designator of *D* is INTERVAL, then

Case:

a) If *D* is specified with a <single datetime field>, then let *ILFP* be the value of <interval leading field precision> and let ***XILFP*** be the canonical XML Schema literal denoting *ILFP* in the lexical representation of XML Schema type **xsd:integer**.

Case:

i) If SECOND was specified in the <single datetime field>, then:

1) Let *IFSP* be the value of <interval fractional seconds precision>. Let ***XIFSP*** be the canonical XML Schema literal denoting IFSP in the lexical representation of XML Schema type **xsd:integer**.

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

`INTERVAL_SECOND_`*`XILFP_XIFSP`*

ii) Otherwise:

1) Let *FT* be the value of <non-second primary datetime field>.

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

`INTERVAL_`*`FT_XILFP`*

b) Otherwise:

i) Let *SFT* be the value of <non-second primary datetime field> in <start field> of *D*.

ii) Let *ILFP* be the value of <interval leading field precision> in <start field> of *D*. Let ***XILFP*** be the canonical XML Schema literal denoting *ILFP* in the lexical representation of XML Schema type **xsd:integer**.

iii) Case:

1) If <end field> of *D* specifies SECOND, then:

A) Let *IFSP* be the value of <interval fractional seconds precision>in <end field> of *D*. Let ***XIFSP*** be the canonical XML Schema literal denoting *IFSP* in the lexical representation of XML Schema type **xsd:integer**.

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

`INTERVAL_`*`SFT_XILFP`*`_SECOND_`*`XIFSP`*

2) Otherwise:

A) Let *EFT* be the value of <non-second primary datetime field> in <end field> of *D*.

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

`INTERVAL_`*`SFT_XILFP_EFT`*

19) 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 7.2, "Mapping a multi-part SQL name to an XML name", to 'Domain', *C*, *S*, and *N*.

20) 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***.

21) 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 7.2, "Mapping a multi-part SQL name to an XML name", to 'UDT', *C*, *S*, and *N*.

22) 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 literal denoting *M* in the lexical representation of XML Schema type **xsd:integer**. Let ***XMLN*** be **Array_*MLIT*.*XMLET***.

23) 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***.

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

Conformance Rules

None.

7.10 Mapping a collection of SQL data types to XML schema data types

Function

Define the mapping of a collection of SQL data types (possibly including domains) to XML Schema data types.

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 *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*.

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

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

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

   a) Let D_i be the *i*-th SQL data type provided for an application of this subclause.

   b) Let $\textbf{\textit{XMLN}}_i$ be the result of applying the mapping defined in Subclause 7.9, "Mapping an SQL data type to an XML name", to D_i.

   c) Let $\textbf{\textit{XMLT}}_i$ be the XML Schema data type that is the result of applying the mapping defined in Subclause 7.11, "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`.

   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 $\textbf{\textit{XMLN}}_i$ is not equivalent to the value of any XML Name in ***XMLTL***, then:

      i) Append $\textbf{\textit{XMLT}}_i$ to ***XMLD***.

      ii) Append $\textbf{\textit{XMLN}}_i$ to ***XMLTL***.

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

Conformance Rules

None.

7.11 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.

General Rules

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

2) 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).

3) If *D* is a character string data 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 *CS* to strings of Unicode. Let *MAXCSL* be the maximum length in characters of Unicode of *CSM*(*S*), for all strings *S* of length *N* of characters of *SQLCS*.

   d) Let ***MLIT*** be the canonical XML Schema literal denoting *MAXCSL* in the lexical representation of XML Schema type **xsd:integer**.

   e) Case:

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

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

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

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

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

   a) Let ***XMLN*** be the result of applying the mapping defined in Subclause 7.9, “Mapping an SQL data type to an XML name”, to *D*.

   b) Let ***XMLT*** be the XML Schema data type that is the result of applying the mapping defined in Subclause 7.15, “Mapping SQL data types to XML schema data types”, to *D* using *NULLS* as the choice of whether to to map null values to absent elements or elements that are marked with **xsi:nil="true"**.

c) Case:

i) If ***XMLT*** is of the form **<xsd:complexType>*XMLTC*</xsd:complexType>**, then let ***SQLCDT*** be the following:

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

ii) If ***XMLT*** is of the form **<xsd:simpleType>*XMLTC*</xsd:simpleType>**, then let ***SQLCDT*** be the following:

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

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

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

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

Conformance Rules

None.

7.12 Mapping an SQL table to an XML element

Function

Define the mapping of an SQL table to an XML element.

General Rules

1) Let *T* be the table provided for an application of this mapping. 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). Let *U* be the authorization identifier that is invoking this mapping.

2) Let *TN* be the <qualified identifier> of the <table name> of *T*.

3) Let ***XMLTN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *TN* using the fully escaped variant of the mapping.

4) Let *n* be the number of rows of *T* and let *m* be the number of visible columns of *T* for *U*.

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

   a) Let R_i be the *i*-th row of *T*.

   b) For *j* ranging from 1 (one) to *m*:

      i) Let C_j be the *j*-th visible column of *T* for *U*.

      ii) Let CN_j be the <column name> of C_j.

      iii) Let $\textbf{\textit{XMLCN}}_j$ be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to CN_j using the fully escaped variant of the mapping.

      iv) Let V_j be the value of C_j.

      v) Case:

         1) If V_j is the null value and *NULLS* is absent, then $\textbf{\textit{XMLC}}_j$ is the zero-length string.

         2) If V_j is the null value and *NULLS* is nil, then $\textbf{\textit{XMLC}}_j$ is

            `<XMLCN`$_j$ `xsi:nil="true" />`

         3) Otherwise:

            A) Let $\textbf{\textit{XMLV}}_j$ be the result of applying the mapping defined in Subclause 7.16, "Mapping SQL data values to XML", to V_j.

            B) $\textbf{\textit{XMLC}}_j$ is

               `<XMLCN`$_j$`>XMLV`$_j$`</XMLCN`$_j$`>`

c) Let ***XMLR***$_i$ be

```
<row>
   XMLC1
   ...
   XMLCm
</row>
```

6) Let ***XMLTE*** be:

```
<XMLTN>
   XMLR1
   ...
   XMLRn
</XMLTN>
```

7) ***XMLTE*** is the XML element that is the result of the application of this clause.

Conformance Rules

None.

7.13 Mapping an SQL schema to an XML element

Function

Define the mapping of an SQL schema to an XML element.

General Rules

1) Let *S* be the schema provided for an application of this mapping. 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). Let *U* be the authorization identifier that is invoking this mapping.

2) Let *SN* be the <unqualified schema name> of *S*.

3) Let ***XMLSN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *SN* using the fully escaped variant of the mapping.

4) Let *n* be the number of visible tables of *S* for *U*.

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

   a) Let T_i be the *i*-th visible table of *S* for *U*.

   b) Let $\mathbf{XMLT}_i$ be the result of applying the mapping defined in Subclause 7.12, "Mapping an SQL table to an XML element", to T_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 *U* as the invoker of this mapping.

6) Let ***XMLSE*** be:

```
<XMLSN>
   XMLT1
   ...
   XMLTn
</XMLSN>
```

7) ***XMLSE*** is the XML element that is the result of the application of this clause.

Conformance Rules

None.

7.14 Mapping an SQL catalog to an XML element

Function

Define the mapping of an SQL catalog to an XML element.

General Rules

1) Let *C* be the catalog provided for an application of this mapping. 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). Let *U* be the authorization identifier that is invoking this mapping.

2) Let *CN* be the <catalog name> of *C*.

3) Let ***XMLCN*** be the result of applying the mapping defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to *CN* using the fully escaped variant of the mapping.

4) Let *n* be the number of visible schemas of *C* for *U*.

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

   a) Let S_i be the *i*-th visible schema of *C* for *U*.

   b) Let $\textbf{\textit{XMLS}}_i$ be the result of applying the mapping defined in Subclause 7.13, "Mapping an SQL schema to an XML element", to S_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 *U* as the invoker of this mapping.

6) Let ***XMLCE*** be:

```
<XMLCN>
   XMLS1
   ...
   XMLSn
</XMLCN>
```

7) ***XMLCE*** is the XML element that is the result of the application of this clause.

Conformance Rules

None.

7.15 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.

General Rules

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

2) 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).

   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 *IM* be the mapping of SQL <identifier>s to XML defined in Subclause 7.1, "Mapping SQL <identifier>s to XML names".

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

5) Let **xsd:** be the XML namespace prefix to be used to identify the XML Schema namespace as shown in Table 2, "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, "Namespace prefixes and their URIs".

Editor's Note
The value of the **sqlxml:** namespace identifier may change. See Possible Problem **XML-001** in the Editor's Notes.

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 Unicode of *CSM*(*S*), for all strings *S* of length *N* of characters of *SQLCS*.

      iii) Let ***NLIT*** and ***MLIT*** be XML Schema literals denoting *N* and *MAXCSL*, resepectively, in the lexical representation of XML Schema type **xsd:integer**.

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:

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

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

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

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

```
name="CHAR"
```

C) It is implementation-dependent 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:

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

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

```
name="VARCHAR"
```

C) It is implementation-dependent 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:

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

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

```
name="CLOB"
```

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

```
collation="SQLCONLIT"
```

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

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

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

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

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

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

ii) It is implementation-dependent whether to encode a binary string in hex or base64.

Case:

1) If the encoding is 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:

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

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

```
name="BLOB"
```

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

```
maxLength="NLIT"
```

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

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

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

```
<xsd:simpleType>
  ANN
  <xsd:restriction base="xsd:EN">
    FACET
  </xsd:restriction>
</xsd: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 **xsd:integer**. Let ***FACETP*** be the XML text:

```
<xsd: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 **xsd:integer**. Let ***FACETS*** be the XML text:

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

iii) Case:

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

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

```
name="NUMERIC"
```

B) It is implementation-dependent 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-dependent 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 **xsd:integer**. It is implementation-dependent whether the XML text ***ANNP*** is the zero-length string or:

```
userPrecision="UPLIT"
```

NOTE 12 – *UP* may be less than *P*, as specified in Syntax Rule 20) of Subclause 6.1, "<data type>", in ISO/IEC 9075-2.

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

```
precision="SLIT"
```

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

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

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

```
<xsd:simpleType>
  ANN
  <xsd:restriction base="xsd:decimal">
    FACETP
    FACETS
  </xsd:restriction>
</xsd: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 **xsd:integer**. Let ***FACETMAX*** be the XML text:

```
<xsd: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 **xsd:integer**. Let ***FACETMIN*** be the XML text:

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

iii) Case:

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

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

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

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

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

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

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

```
<xsd:simpleType>
  ANN
  <xsd:restriction base="xsd:integer">
    FACETMAX
    FACETMIN
  </xsd:restriction>
</xsd: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-dependent 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-dependent whether the XML text ***ANNT*** is the zero-length string or:

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

3) Otherwise:

A) It is implementation-dependent 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 **`xsd:integer`**. It is implementation-dependent whether the XML text ***ANNUP*** is the zero-length string or:

```
userPrecision="UPLIT"
```

NOTE 13 – *UP* may be less than *P*, as specified in Syntax Rule 20) of Subclause 6.1, "<data type>", in ISO/IEC 9075-2.

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

```
maxExponent="MAXLIT"
```

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

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

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

```
<xsd:simpleType>
  ANN
  <xsd:restriction base="xsd:TYPE">
  </xsd:restriction>
</xsd:simpleType>
```

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

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

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

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

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

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

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

ii) Case:

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

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

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

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

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

```
name="TIME"
```

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

```
scale="SLIT"
```

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

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

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

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

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

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

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

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

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

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

```
scale="SLIT"
```

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

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

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

```
<xsd:simpleType>
  ANN
  <xsd:restriction base="xsd:time">
    FACETP
  </xsd:restriction>
</xsd: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 **xsd:integer**.

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

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

iii) Case:

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

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

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

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

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

```
name="TIMESTAMP"
```

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

```
scale="SLIT"
```

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

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

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

```
<xsd:simpleType>
  ANN
  <xsd:restriction base="xsd:dateTime">
    FACETP
  </xsd:restriction>
</xsd: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 **xsd:integer**.

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

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

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

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

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

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

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

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

```
scale="SLIT"
```

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

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

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

```
<xsd:simpleType>
  ANN
  <xsd:restriction base="xsd:dateTime">
    FACETP
  </xsd:restriction>
</xsd: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 **xsd:integer**. It is implementation-dependent 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 **xsd:integer**. Let the XML text ***SECS*** be:

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

It is implementation-dependent 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:

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

B) It is implementation-dependent 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:

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

B) It is implementation-dependent 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:

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

B) It is implementation-dependent 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:

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

B) It is implementation-dependent 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:

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

B) It is implementation-dependent 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:

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

B) It is implementation-dependent 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:

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

B) It is implementation-dependent 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:

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

B) It is implementation-dependent 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:

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

B) It is implementation-dependent 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:

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

B) It is implementation-dependent 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:

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

B) It is implementation-dependent 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:

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

B) It is implementation-dependent 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:

```
<xsd:pattern value="-?PTSECSS"/>
```

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

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

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

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

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

```
<xsd:simpleType>
  ANN
  <xsd:restriction base="xsd:duration">
    FACETP
  </xsd:restriction>
</xsd: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 7.9, “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-dependent whether the XML text ***ANN*** is the zero-length string or given by:

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

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

```
<xsd:simpleType>
  ANN
  <xsd:restriction base='XMLN'/>
</xsd: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 $\textbf{\textit{XMLMT}}_i$ be the result of applying the mapping in Subclause 7.9, "Mapping an SQL data type to an XML name", to FT_i, for *i* between 1 (one) and *N*.

iii) Let $\textbf{\textit{XMLFN}}_i$ be the result of applying the mapping in Subclause 7.9, "Mapping an SQL data type to an XML name", FN_i, for *i* between 1 (one) and *N*.

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

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

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

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

```
<xsd:complexType>
  ANN
  <xsd:sequence>
    <xsd:element name='XMLFN1' type='XMLMT1' XMLNULLS/>
    . . .
    <xsd:element name='XMLFNN' type='XMLMTN' XMLNULLS/>
  </xsd:sequence>
</xsd: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 7.9, "Mapping an SQL data type to an XML name", to *ST*.

iii) Let *DTC*, *DTS*, and *DTN* be the distinct type's catalog name, schema name, and type name, respectively.

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

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

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

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

```
<xsd:simpleType>
  ANN
  <xsd:restriction base='XMLN'/>
</xsd: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 7.9, "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 **xsd:integer**.

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

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

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

```
<xsd:complexType>
  ANN
  <xsd:sequence>
    <xsd:element name='element' minOccurs='0'
                    maxOccurs='MLIT' nillable='true'
                    type='XMLN'/>
  </xsd:sequence>
</xsd: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 7.9, "Mapping an SQL data type to an XML name", to *ET*.

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

```
<xsd:annotation>
  <xsd:appinfo>
    <sqlxml:sqltype kind='MULTISET'
                       mappedElementType='XMLN'/>
  </xsd:appinfo>
</xsd:annotation>
```

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

```
<xsd:complexType>
  ANN
  <xsd:sequence>
    <xsd:element name='element' minOccurs='0'
                    maxOccurs='unbounded' nillable='true'
                    type='XMLN'/>
  </xsd:sequence>
</xsd:complexType>
```

9) ***XMLT*** defines the mapping of *SQLT* into XML.

Conformance Rules

None.

7.16 Mapping SQL data values to XML

Function

Define the mapping of nonnull SQL data values to XML.

Syntax Rules

None.

General Rules

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

   Case:

   a) If *DV* is a value of a domain, then let *SQLT* be the data type of that domain, and let *SQLV* be *DV*.

   b) 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)
      ```

   c) Otherwise, let *SQLT* be the SQL data 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 whether to map them to elements that are marked with **xsi:nil="true"** (nil).

3) Let ***XMLT*** be the XML Schema type obtained by mapping *SQLT* using the Rules of Subclause 7.15, "Mapping SQL data types to XML schema data types".

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

5) Let *CV* be the result of

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

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

7) Case:

   a) If *SQLT* is a character string type, then let *CS* be the character set of *SQLT*. Let ***XMLV*** be the result of mapping *SQLV* to Unicode using the implementation-defined mapping of character strings of *CS* to Unicode.

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

      Case:

      i) If ***XMLT*** encodes a binary string in hex, then let ***XMLV*** be the hex encoding of *SQLV*.

      ii) Otherwise, let ***XMLV*** be the base64 encoding 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) 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) 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) For *i* between 1 (one) and *N*, let $\textbf{\textit{XMLFN}}_i$ be the result of applying the mapping in Subclause 7.1, "Mapping SQL <identifier>s to XML names", to FN_i, using the fully escaped variant of the mapping.

2) Case:

A) If FV_i is the null value and *NULLS* is absent, then let $\textbf{\textit{XMLE}}_i$ be the empty string.

B) If FV_i is the null value and *NULLS* is nil, then let $\textbf{\textit{XMLE}}_i$ be the XML element:

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

C) Otherwise, let the XML text $\textbf{\textit{XMLV}}_i$ be the result of applying the mapping defined in this Subclause to FV_i. Let $\textbf{\textit{XMLE}}_i$ be the XML element:

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

3) Let ***XMLV*** be the sequence of XML elements:

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

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 null, then let the XML text $\textbf{\textit{XMLE}}_i$ 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 the sequence of XML elements:

$XMLE_1$ $XMLE_2$ ... $XMLE_N$

8) ***XMLV*** is the result of mapping *SQLV* to XML.

Editor's Note These rules do not handle the escaping of the reserved symbols such as <less than operator>, which might be done using either entities (such as **<**) or by escaping the entire string using **CDATA**. See Possible Problem **XML-005** in the Editor's Notes.

Conformance Rules

None.

7.17 Mapping XML names to SQL <identifier>s

Function

Define the mapping of XML Names to SQL <identifier>s.

Syntax Rules

None.

General Rules

1) Let ***XMLN*** be an XML Name in an application of this Subclause. ***XMLN*** is a sequence of Unicode characters. Let N be the number of characters in ***XMLN***. Let $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, . . . , $\boldsymbol{X}_N$ be the characters of ***XMLN*** in order from left to right.

2) Let the N Unicode character strings U_1, U_2, . . . , U_N be defined as follows:

   If U_i, 1 (one) $\leq i \leq N$, has not yet been determined, then

   Case:

   a) If $\boldsymbol{X}_{\mathrm{i}}$ = '_' (an <underscore>), and $\boldsymbol{X}_{\mathrm{i+1}}$ = 'x', and each of $\boldsymbol{X}_{\mathrm{i+2}}$, $\boldsymbol{X}_{\mathrm{i+3}}$, $\boldsymbol{X}_{\mathrm{i+4}}$, and $\boldsymbol{X}_{\mathrm{i+5}}$ are all <hexit>s, and $\boldsymbol{X}_{\mathrm{i+6}}$ = '_', then

      Case:

      i) If i = 1 (one) and $\boldsymbol{X}_3$, $\boldsymbol{X}_4$, $\boldsymbol{X}_5$, and $\boldsymbol{X}_6$ are all 'F', then let U_1, U_2, U_3, U_4, U_5, U_6, and U_7 be the zero-length string.

      ii) If the Unicode codepoint U+$X_{i+2}X_{i+3}X_{i+4}X_{i+5}$ is a valid Unicode character UC, then let U_i be the character string of length 1 (one) whose character is UC and let $\boldsymbol{U}_{\mathrm{i+1}}$, $\boldsymbol{U}_{\mathrm{i+2}}$, $\boldsymbol{U}_{\mathrm{i+3}}$, $\boldsymbol{U}_{\mathrm{i+4}}$, $\boldsymbol{U}_{\mathrm{i+5}}$, and $\boldsymbol{U}_{\mathrm{i+6}}$ be the zero-length string.

      iii) Otherwise, $\boldsymbol{U}_{\mathrm{i}}$, $\boldsymbol{U}_{\mathrm{i+1}}$, $\boldsymbol{U}_{\mathrm{i+2}}$, $\boldsymbol{U}_{\mathrm{i+3}}$, $\boldsymbol{U}_{\mathrm{i+4}}$, $\boldsymbol{U}_{\mathrm{i+5}}$, and $\boldsymbol{U}_{\mathrm{i+6}}$ are implementation-defined.

   b) If $\boldsymbol{X}_{\mathrm{i}}$ = '_' (an <underscore>), and $\boldsymbol{X}_{\mathrm{i+1}}$ = 'x', and each of $\boldsymbol{X}_{\mathrm{i+2}}$, $\boldsymbol{X}_{\mathrm{i+3}}$, $\boldsymbol{X}_{\mathrm{i+4}}$, $\boldsymbol{X}_{\mathrm{i+5}}$, $\boldsymbol{X}_{\mathrm{i+6}}$, $\boldsymbol{X}_{\mathrm{i+7}}$, $\boldsymbol{X}_{\mathrm{i+8}}$, and $\boldsymbol{X}_{\mathrm{i+9}}$, are all <hexit>s, and $\boldsymbol{X}_{\mathrm{i+10}}$ = '_', then

      Case:

      i) If the Unicode codepoint U+$X_{i+2}X_{i+3}X_{i+4}X_{i+5}$ $X_{i+6}X_{i+7}X_{i+8}X_{i+9}$ is a valid Unicode character UC, then let $\boldsymbol{U}_{\mathrm{i}}$ be the character string of length 1 (one) whose character is UC and let $\boldsymbol{U}_{\mathrm{i+1}}$, $\boldsymbol{U}_{\mathrm{i+2}}$, $\boldsymbol{U}_{\mathrm{i+3}}$, $\boldsymbol{U}_{\mathrm{i+4}}$, $\boldsymbol{U}_{\mathrm{i+5}}$, $\boldsymbol{U}_{\mathrm{i+6}}$, $\boldsymbol{U}_{\mathrm{i+7}}$, $\boldsymbol{U}_{\mathrm{i+8}}$, $\boldsymbol{U}_{\mathrm{i+9}}$, and $\boldsymbol{U}_{\mathrm{i+10}}$, be the zero-length string.

      ii) Otherwise, $\boldsymbol{U}_{\mathrm{i}}$, $\boldsymbol{U}_{\mathrm{i+1}}$, $\boldsymbol{U}_{\mathrm{i+2}}$, $\boldsymbol{U}_{\mathrm{i+3}}$, $\boldsymbol{U}_{\mathrm{i+4}}$, $\boldsymbol{U}_{\mathrm{i+5}}$, $\boldsymbol{U}_{\mathrm{i+6}}$, $\boldsymbol{U}_{\mathrm{i+7}}$, $\boldsymbol{U}_{\mathrm{i+8}}$, $\boldsymbol{U}_{\mathrm{i+9}}$, and $\boldsymbol{U}_{\mathrm{i+10}}$ are implementation-defined.

   c) Otherwise, let U_i be the character string of length 1 (one) whose character is X_i.

3) Let U be the Unicode character string constructed by concatenating every U_i, 1 (one) $\leq i \leq N$, in order by i.

4) Let *SQLI* be the SQL_TEXT character string obtained by mapping the Unicode character string U to SQL_TEXT using the implementation-defined mapping of Unicode to SQL_TEXT. If *SQLI* can not be mapped to SQL_TEXT, then an exception condition is raised: *SQL/XML mapping error — unmappable XML Name*.

5) The SQL <identifier> that is the mapping of ***XMLN*** is the <delimited identifier> **"*SQLI*"**.

Conformance Rules

None.

8 Additional common rules

This Clause modifies Clause 9, "Additional common rules", in ISO/IEC 9075-2.

8.1 Type precedence list determination

This Subclause modifies Subclause 9.5, "Type precedence list determination", in ISO/IEC 9075-2.

Function

Determine the type precedence list of a given type.

Syntax Rules

1) Insert this SR If *DT* is the XML type, then *TPL* is *DT*.

Conformance Rules

No additional Conformance Rules.

8.2 Type name determination

This Subclause modifies Subclause 9.7, "Type name determination", in ISO/IEC 9075-2.

Function

Determine an <identifier> given the name of a predefined data type.

Syntax Rules

1) Augment SR 2) If *DT* specifies XML, then let *FNSDT* be 'XML'.

Conformance Rules

No additional Conformance Rules.

8.3 Determination of identical values

This Subclause modifies Subclause 9.8, "Determination of identical values", in ISO/IEC 9075-2.

Function

Determine whether two instances of values are identical, that is to say, are occurrences of the same value.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 2)e) If *V1* and *V2* are both of the XML type, then whether *V1* and *V2* are identical or not identical is implementation-defined.

Conformance Rules

No additional Conformance Rules.

8.4 Equality operations

This Subclause modifies Subclause 9.9, "Equality operations", in ISO/IEC 9075-2.

Function

Specify the prohibitions and restrictions by data type on operations that involve testing for equality.

Syntax Rules

1) Insert this SR The declared type of an operand of an equality operation shall not be XML-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

8.5 Grouping operations

This Subclause modifies Subclause 9.10, "Grouping operations", in ISO/IEC 9075-2.

Function

Specify the prohibitions and restrictions by data type on operations that involve grouping of data.

Syntax Rules

1) Insert this SR The declared type of an operand of a grouping operation shall not be XML-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

- 1 paragraph deleted.

No additional Conformance Rules.

8.6 Multiset element grouping operations

This Subclause modifies Subclause 9.11, "Multiset element grouping operations", in ISO/IEC 9075-2.

Function

Specify the prohibitions and restrictions by data type on the declared element type of a multiset for operations that involve grouping the elements of a multiset.

Syntax Rules

1) Insert this SR The declared element type of a multiset operand of a multiset element grouping operation shall not be XML-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

8.7 Ordering operations

This Subclause modifies Subclause 9.12, "Ordering operations", in ISO/IEC 9075-2.

Function

Specify the prohibitions and restrictions by data type on operations that involve ordering of data.

Syntax Rules

1) Insert this SR The declared type of an operand of an ordering operation shall not be XML-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

9 Additional common elements

This Clause modifies Clause 10, "Additional common elements", in ISO/IEC 9075-2.

9.1 <aggregate function>

This Subclause modifies Subclause 10.9, "<aggregate function>", in ISO/IEC 9075-2.

Function

Specify a value computed from a collection of rows.

Format

```
<aggregate function> ::=
         !! All alternatives from ISO/IEC 9075-2
       | <XML aggregate>

<XML aggregate> ::=
       XMLAGG <left paren>
         <XML value expression>
         [ ORDER BY <sort specification list> ]
       <right paren>
```

Syntax Rules

1) Insert this SR The declared type of <XML aggregate> is XML.

Access Rules

No additional Access Rules.

General Rules

1) Insert this SR If <XML aggregate> is specified, then:

   a) If <sort specification list> is specified, then let *N* be the number of <sort key>s; otherwise, let *N* be 0 (zero).

   b) Let *TXA* be the table of *N*+1 columns obtained by applying <XML value expression> to each row of *T1* to obtain the first column of *TXA*, and, for all *i* between 1 (one) and *N*, applying the <value expression> simply contained in the *i*-th <sort key> to each row of *T1* to obtain the (*i*+1)-th column of *TXA*.

   c) Every row of *TXA* in which the value of the first column is the null value is removed from *TXA*.

   d) Let *TXA* be ordered according to the values of the <sort key>s found in the second through *N*-th columns of *TXA*. If *N* is 0 (zero), then the ordering of *TXA* is implementation-dependent.

e) Case:

i) If *TXA* is empty, then the result of <XML aggregate> is the null value.

ii) If there exist two different rows in *TXA* such that the value of the first column in one row is an XML document, and the value of the other row is not an empty XML value, then an exception condition is raised: *data exception — concatenation with an XML document*.

iii) Otherwise, the result is the concatenation of the values in the first column of *TXA*, in the order of rows of *TXA*.

Conformance Rules

1) Insert this CR Without Feature X001, "Elementary XML type", an <aggregate function> shall not be an <XML aggregate>.

2) Insert this CR Without Feature X002, "Advanced XML type", <XML aggregate> shall not specify <sort specification list>.

9.2 XQuery syntactic elements

Function

Reference rules of *XQuery 1.0: An XML Query Language*.

Format

```
<XQuery ExprSequence> ::=
       !! See rule 3 in XQuery 1.0: An XML Query Language

<XQuery Constructor> ::=
       !! See rule 24 in XQuery 1.0: An XML Query Language

<XQuery EnclosedExpr> ::=
       !! See rule 67 in XQuery 1.0: An XML Query Language

<XQuery Variable> ::=
       !! See rule 158 in XQuery 1.0: An XML Query Language

<XQuery QName> ::=
       !! See rule 189 in XQuery 1.0: An XML Query Language
```

Editor's Note
XQuery rule names and numbers shall be tracked against changes in XQuery 1.0. See Possible Problem **XML-014** in the Editor's Notes.

Syntax Rules

None.

Access Rules

None.

General Rules

1) The result of an <XQuery Constructor> *XC* for a particular binding of values to the <XQuery Variable>s contained in *XC* is determined by the rules of *XQuery 1.0: An XML Query Language*.

Conformance Rules

None.

10 SQL-client modules

This Clause modifies Clause 13, "SQL-client modules", in ISO/IEC 9075-2.

10.1 Calls to an <externally-invoked procedure>

This Subclause modifies Subclause 13.4, "Calls to an <externally-invoked procedure>", in ISO/IEC 9075-2.

Function

Define the call to an <externally-invoked procedure> by an SQL-agent.

Syntax Rules

1) Insert into SR2)e)

```
DATA_EXCEPTION_INVALID_XML_VALUE_IN_XML_OPERATION:
        constant SQLSTATE_TYPE :="2200J";
DATA_EXCEPTION_CONCATENATION_WITH_AN_XML_DOCUMENT:
        constant SQLSTATE_TYPE :="2200K";
DATA_EXCEPTION_INVALID_XQUERY_CONSTRUCTOR:
        constant SQLSTATE_TYPE :="2200L";
DATA_EXCEPTION_XMLGEN_EVALUATION_FAILURE:
        constant SQLSTATE_TYPE :="2200M";
DATA_EXCEPTION_XML_VALUE_NOT_WELL_FORMED:
        constant SQLSTATE_TYPE :="2200N";
SQLXML_MAPPING_ERROR_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="0N000";
SQLXML_MAPPING_ERROR_UNMAPPABLE_XML_NAME:
        constant SQLSTATE_TYPE :="0N001";
```

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

- 1 paragraph deleted.

No additional Conformance Rules.

10.2 Data type correspondences

This Subclause modifies Subclause 13.6, "Data type correspondences", in ISO/IEC 9075-2.

Function

Specify the data type correspondences for SQL data types and host language types.

Tables

Table 3, "Data type correspondences for Ada", modifies Table 16, "Data type correspondences for Ada", in ISO/IEC 9075-2.

Table 3—Data type correspondences for Ada

SQL Data Type	Ada Data Type
All alternatives from ISO/IEC 9075-2	
XML	*None*

Table 4, "Data type correspondences for C", modifies Table 17, "Data type correspondences for C", in ISO/IEC 9075-2.

Table 4—Data type correspondences for C

SQL Data Type	C Data Type
All alternatives from ISO/IEC 9075-2	
XML	*None*

Table 5, "Data type correspondences for COBOL", modifies Table 18, "Data type correspondences for COBOL", in ISO/IEC 9075-2.

Table 5—Data type correspondences for COBOL

SQL Data Type	COBOL Data Type
All alternatives from ISO/IEC 9075-2	
XML	*None*

Table 6, "Data type correspondences for Fortran", modifies Table 19, "Data type correspondences for Fortran", in ISO/IEC 9075-2.

Table 6—Data type correspondences for Fortran

SQL Data Type	Fortran Data Type
All alternatives from ISO/IEC 9075-2	

Table 6—Data type correspondences for Fortran (Cont.)

SQL Data Type	Fortran Data Type
XML	*None*

Table 7, "Data type correspondences for MUMPS", modifies Table 20, "Data type correspondences for MUMPS", in ISO/IEC 9075-2.

Table 7—Data type correspondences for MUMPS

SQL Data Type	MUMPS Data Type
All alternatives from ISO/IEC 9075-2	
XML	*None*

Table 8, "Data type correspondences for Pascal", modifies Table 21, "Data type correspondences for Pascal", in ISO/IEC 9075-2.

Table 8—Data type correspondences for Pascal

SQL Data Type	Pascal Data Type
All alternatives from ISO/IEC 9075-2	
XML	*None*

Table 9, "Data type correspondences for PL/I", modifies Table 22, "Data type correspondences for PL/I", in ISO/IEC 9075-2.

Table 9—Data type correspondences for PL/I

SQL Data Type	PL/I Data Type
All alternatives from ISO/IEC 9075-2	
XML	*None*

Conformance Rules

No additional Conformance Rules.

11 Dynamic SQL

This Clause modifies Clause 19, "Dynamic SQL", in ISO/IEC 9075-2.

11.1 Description of SQL descriptor areas

This Subclause modifies Subclause 19.1, "Description of SQL descriptor areas", in ISO/IEC 9075-2.

Function

Specify the identifiers, data types, and codes used in SQL item descriptor areas.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 1) Table 10, “Codes used for SQL data types in Dynamic SQL”, specifies the codes associated with the SQL data types.

Table 10, “Codes used for SQL data types in Dynamic SQL”, modifies Table 25, "Codes used for SQL data types in Dynamic SQL", in ISO/IEC 9075-2.

Table 10—Codes used for SQL data types in Dynamic SQL

Data Type	Code
All alternatives from ISO/IEC 9075-2	*All alternatives from ISO/IEC 9075-2*
XML	137

Conformance Rules

- 1 paragraph deleted.

No additional Conformance Rules.

12 The SQL/XML XML schema

12.1 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"?>
<xsd:schema
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.iso-standards.net/9075/2001/12/sqlxml"
      xmlns:sqlxml="http://www.iso-standards.net/9075/2001/12/sqlxml">
  <xsd:annotation>
    <xsd:documentation>
      ISO/IEC 9075-14:2003 (SQL/XML)
      This document contains definitions of types and
      annotations as specified in ISO/IEC 9075-14:200n.
    </xsd:documentation>
  </xsd:annotation>

  <xsd:simpleType name="kindKeyword">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="PREDEFINED"/>
      <xsd:enumeration value="DOMAIN"/>
      <xsd:enumeration value="ROW"/>
      <xsd:enumeration value="DISTINCT"/>
      <xsd:enumeration value="ARRAY"/>
      <xsd:enumeration value="MULTISET"/>
    </xsd:restriction>
  </xsd:simpleType>
```

```
<xsd:simpleType name="typeKeyword">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="CHAR"/>
    <xsd:enumeration value="VARCHAR"/>
    <xsd:enumeration value="CLOB"/>
    <xsd:enumeration value="BLOB"/>
    <xsd:enumeration value="NUMERIC"/>
    <xsd:enumeration value="DECIMAL"/>
    <xsd:enumeration value="INTEGER"/>
    <xsd:enumeration value="SMALLINT"/>
    <xsd:enumeration value="BIGINT"/>
    <xsd:enumeration value="FLOAT"/>
    <xsd:enumeration value="REAL"/>
    <xsd:enumeration value="DOUBLE PRECISION"/>
    <xsd:enumeration value="BOOLEAN"/>
    <xsd:enumeration value="DATE"/>
    <xsd:enumeration value="TIME"/>
    <xsd:enumeration value="TIME WITH TIME ZONE"/>
    <xsd:enumeration value="TIMESTAMP"/>
    <xsd:enumeration value="TIMESTAMP WITH TIME ZONE"/>
    <xsd:enumeration value="INTERVAL YEAR"/>
    <xsd:enumeration value="INTERVAL YEAR TO MONTH"/>
    <xsd:enumeration value="INTERVAL MONTH"/>
    <xsd:enumeration value="INTERVAL DAY"/>
    <xsd:enumeration value="INTERVAL DAY TO HOUR"/>
    <xsd:enumeration value="INTERVAL DAY TO MINUTE"/>
    <xsd:enumeration value="INTERVAL DAY TO SECOND"/>
    <xsd:enumeration value="INTERVAL HOUR"/>
    <xsd:enumeration value="INTERVAL HOUR TO MINUTE"/>
    <xsd:enumeration value="INTERVAL HOUR TO SECOND"/>
    <xsd:enumeration value="INTERVAL MINUTE"/>
    <xsd:enumeration value="INTERVAL MINUTE TO SECOND"/>
    <xsd:enumeration value="INTERVAL SECOND"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="fieldType">
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="mappedType" type="xsd:string"/>
</xsd:complexType>
```

```
<xsd:element name="sqltype">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="field" type="fieldType"
                   minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="kind"
                   type="sqlxml:kindKeyword"/>
    <xsd:attribute name="name"
                   type="sqlxml:typeKeyword" use="optional"/>
    <xsd:attribute name="length" type="xsd:integer"
                   use="optional"/>
    <xsd:attribute name="maxLength" type="xsd:integer"
                   use="optional"/>
    <xsd:attribute name="characterSetName" type="xsd:string"
                   use="optional"/>
    <xsd:attribute name="collation" type="xsd:string"
                   use="optional"/>
    <xsd:attribute name="precision" type="xsd:integer"
                   use="optional"/>
    <xsd:attribute name="scale" type="xsd:integer"
                   use="optional"/>
    <xsd:attribute name="maxExponent" type="xsd:integer"
                   use="optional"/>
    <xsd:attribute name="minExponent" type="xsd:integer"
                   use="optional"/>
    <xsd:attribute name="userPrecision" type="xsd:integer"
                   use="optional"/>
    <xsd:attribute name="leadingPrecision" type="xsd:integer"
                   use="optional"/>
    <xsd:attribute name="maxElements" type="xsd:integer"
                   use="optional"/>
    <xsd:attribute name="catalogName" type="xsd:string"
                   use="optional"/>
    <xsd:attribute name="schemaName" type="xsd:string"
                   use="optional"/>
    <xsd:attribute name="domainName" type="xsd:string"
                   use="optional"/>
    <xsd:attribute name="typeName" type="xsd:string"
                   use="optional"/>
    <xsd:attribute name="mappedType" type="xsd:string"
                   use="optional"/>
    <xsd:attribute name="mappedElementType" type="xsd:string"
                   use="optional"/>
    <xsd:attribute name="final" type="xsd:boolean"
                   use="optional"/>
  </xsd:complexType>
</xsd:element>

<xsd:simpleType name="objectType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="CATALOG" />
      <xsd:enumeration value="SCHEMA" />
      <xsd:enumeration value="BASE TABLE" />
      <xsd:enumeration value="VIEWED TABLE" />
      <xsd:enumeration value="CHARACTER SET" />
      <xsd:enumeration value="COLLATION" />
   </xsd:restriction>
</xsd:simpleType>
```

```
  <xsd:element name="sqlname">
     <xsd:complexType>
        <xsd:attribute name="type" type="sqlxml:objectType"
                          use="required" />
        <xsd:attribute name="catalogName" type="xsd:string" />
        <xsd:attribute name="schemaName" type="xsd:string" />
        <xsd:attribute name="localName" type="xsd:string" />
     </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Editor's Note
The value of the **sqlxml:** namespace identifier may change. See Possible Problem **XML-001**.

Editor's Note
The publication date of this part shall be supplied in the annotation in the **sqlxml:** namespace in two places. See Possible Problem **XML-003**.

General Rules

None.

Editor's Note
Which is normative, Clause 12, "The SQL/XML XML schema", or the actual URL that defines this namespace on-line? If it is the URL, should this clause be downgraded to an informative Annex? If the clause or annex is modified by a TC, is the URL or its contents also changed? See Possible Problem **XML-004** in the Editor's Notes.

Conformance Rules

None.

13 Definition Schema

This Clause modifies Clause 6, "Definition Schema", in ISO/IEC 9075-11.

13.1 DATA_TYPE_DESCRIPTOR base table

This Subclause modifies Subclause 6.19, "DATA_TYPE_DESCRIPTOR base table", in ISO/IEC 9075-11.

Function

The DATA_TYPE_DESCRIPTOR table has one row for each domain, one row for each column (in each table) and for each attribute (in each structured type) that is defined as having a data type rather than a domain, one row for each distinct type, one row for the result type of each SQL parameter of each SQL-invoked routine, one row for the result type of each method specification, one row for each parameter of each method specification, and one row for each structured type whose associated reference type has a user-defined representation. It effectively contains a representation of the data type descriptors.

Definition

Augment constraint DATA_TYPE_DESCRIPTOR_DATA_TYPE_CHECK_COMBINATIONS:

Alter the final OR clause of the constraint:

```
OR
  ( DATA_TYPE NOT IN
    ( 'CHARACTER', 'CHARACTER VARYING', 'CHARACTER LARGE OBJECT',
      'BINARY LARGE OBJECT',
      'NUMERIC', 'DECIMAL', 'SMALLINT', 'INTEGER', 'BIGINT',
      'FLOAT', 'REAL', 'DOUBLE PRECISION',
      'DATE', 'TIME', 'TIMESTAMP',
      'INTERVAL', 'BOOLEAN', 'USER-DEFINED',
      'REF', 'ROW', 'ARRAY', 'MULTISET', 'XML' ) ) ),
```

Add the following OR clause to the end of the constraint:

```
OR
  ( DATA_TYPE = 'XML'
  AND
    ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
      NUMERIC_SCALE, DATETIME_PRECISION, CHARACTER_OCTET_LENGTH,
      CHARACTER_MAXIMUM_LENGTH, INTERVAL_TYPE, INTERVAL_PRECISION )
        IS NULL
  AND
    ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
      USER_DEFINED_TYPE_NAME ) IS NULL
  AND
    ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
  AND
    MAXIMUM_CARDINALITY IS NULL )
```

Description

1) Insert this Description If DATA_TYPE is 'XML', then the data type being described is the XML type.

14 Status codes

This Clause modifies Clause 23, "Status codes", in ISO/IEC 9075-2.

14.1 SQLSTATE

This Subclause modifies Subclause 23.1, "SQLSTATE", in ISO/IEC 9075-2.

Table 11, "SQLSTATE class and subclass values", modifies Table 32, "SQLSTATE class and subclass values", in ISO/IEC 9075-2.

Table 11—SQLSTATE class and subclass values

Category	Condition	Class	Subcondition	Subclass
X	data exception	22	(*no subclass*)	000
			invalid XML value in XML operation	00J
			concatenation with an XML document	00K
			invalid XQuery constructor	00L
			XMLGEN evaluation failure	00M
			XML value not well-formed	00N
X	SQL/XML mapping error	0N	(*no subclass*)	000
			unmappable XML Name	001

15 Conformance

This Clause modifies Clause 24, "Conformance", in ISO/IEC 9075-2.

- to be defined -

Annex A
(informative)

SQL Conformance Summary

This Annex modifies Annex A, "SQL Conformance Summary", in ISO/IEC 9075-2.

- The contents of this Annex will be supplied automatically prior to any FDIS ballot -

Annex B
(informative)

Implementation-defined elements

This Annex modifies Annex B, "Implementation-defined elements", in ISO/IEC 9075-2.

This Annex references those features that are identified in the body of this part of ISO/IEC 9075 as implementation-defined.

The term implementation-defined is used to identify characteristics that may differ between SQL-implementations, but shall be defined.

1) Insert this list element Subclause 4.6.1, "Mapping SQL character sets to Unicode":

   a) The mapping of an SQL character set to Unicode is implementation-defined.

2) Insert this list element Subclause 4.6.9, "Mapping Unicode to SQL character sets":

   a) The mapping of Unicode to a character set in the SQL-environment is implementation-defined.

3) Insert this list element Subclause 7.1, "Mapping SQL <identifier>s to XML names":

   a) If *S* is a character in an SQL <identifier> *SQLI* and *S* has no mapping to Unicode, then the mapping of *S* to create an XML Name corresponding to *SQLI* is implementation-defined.

4) Insert this list element Subclause 7.17, "Mapping XML names to SQL <identifier>s":

   a) The treatment of an escape sequence of the form _x***NNNN***_ or _x***NNNNNNNN***_ whose corresponding Unicode code point U+*NNNN* or U+*NNNNNNNN* is not a valid Unicode character is implementation-defined.

5) Insert this list element Subclause 8.3, "Determination of identical values":

   a) Whether two XML values are identical or not identical is implementation-defined.

Annex C
(informative)

Implementation-dependent elements

This Annex modifies Annex C, "Implementation-dependent elements", in ISO/IEC 9075-2.

This Annex references those features that are identified in the body of this part of ISO/IEC 9075 as implementation-dependent.

The term implementation-dependent is used to identify characteristics that may differ between SQL-implementations, but are not necessarily specified for any particular SQL-implementation.

1) Insert this list element Subclause 7.9, “Mapping an SQL data type to an XML name”:

   a) 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, are mapped to the same XML Name.

2) Insert this list element Subclause 7.15, “Mapping SQL data types to XML schema data types”:

   a) All annotations are implementation-dependent.

   b) It is implementation-dependent whether to encode a binary string in hex or base64.

3) Insert this list element Subclause 7.16, “Mapping SQL data values to XML”:

   a) The ordering of elements of a multiset is implementation-dependent.

4) Insert this list element Subclause 9.1, “<aggregate function>”:

   a) The order in which items are concatenated in the result of XMLAGG is implementation-dependent if no user-specified ordering is specified or if the user-specified ordering is not a total ordering.

Annex D
(informative)

Incompatibilities with ISO/IEC 9075-2:1999

This Annex modifies Annex E, "Incompatibilities with ISO/IEC 9075-2:1999", in ISO/IEC 9075-2.

This edition of this part of ISO/IEC 9075 introduces some incompatibilities with the earlier version of Database Language SQL as specified in ISO/IEC 9075-2:1999.

Except as specified in this Annex, features and capabilities of Database Language SQL are compatible with ISO/IEC 9075-2:1999.

1) Augment list element 10) A number of additional <reserved word>s have been added to the language. These <reserved word>s are:

   — XML

   — XMLAGG

   — XMLATTRIBUTES

   — XMLCONCAT

   — XMLELEMENT

   — XMLFOREST

   — XMLGEN

Annex E
(informative)

SQL feature and package taxonomy

This Annex modifies Annex F, "SQL feature and package taxonomy", in ISO/IEC 9075-2.

- to be supplied -

Index

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

— D —

— E —

— F —

— T —

— U —

— V —

— W —

— X —

— Y —

— Z —

1 Possible problems with SQL/XML

I observe some possible problems with SQL/XML as defined in this document. These are noted below. Further contributions to this list are welcome. Deletions from the list (resulting from change proposals that correct the problems or from research indicating that the problems do not, in fact, exist) are even more welcome. Other comments may appear in the same list.

Because of the highly dynamic nature of this list (problems being removed because they are solved, new problems being added), it has become rather confusing to have the problem numbers automatically assigned by the document production facility. In order to reduce this confusion, I have instead assigned "fixed" numbers to each possible problem. These numbers will not change from printing to printing, but will instead develop "gaps" between numbers as problems are solved.

Possible problems related to SQL/XML

Significant Possible Problems:

999 In the body of the Working Draft, I have occasionally highlighted a point that requires urgent attention thus:

****Editor's Note****
Text of the problem.

These items are indexed under "**Editor's Note**".

XML-001 The following Possible Problem has been noted:

Severity: Major Technical
Reference: P14-nn.nn, Subclause 4.5, "Namespaces"
Note at: The end of Subclause 4.5, "Namespaces", and in the General Rules of Subclause 7.15, "Mapping SQL data types to XML schema data types".
Source: WG3:E3A-003
Possible Problem:

The value of the ***sqlxml:*** namespace must be supplied correctly and finally.

XML-002 The following Possible Problem has been noted:

Severity: Major Technical
Reference: Subclause 2.2, "Publicly-available specifications"
Note at: The end of Subclause 2.2, "Publicly-available specifications"
Source: WG3:E3A-003
Possible Problem:

Many of the normative references have not been finalized. It will be necessary to reference the correct specifications as they become available. This may entail changes to other clauses of this standard to align with the final forms of these specifications.

XML-003 The following Possible Problem has been noted:

Severity: Major Technical
Reference: Subclause 12.1, "The SQL/XML XML schema"
Note at: At the end of the Syntax Rules of Subclause 12.1, "The SQL/XML XML schema"
Source: WG3:E3A-003

Possible Problem:

The publication date of this part must be supplied in the annotation in the ***sqlxml:*** namespace in two places.

XML-004 The following Possible Problem has been noted:

Severity: Major Technical
Reference: Clause 12, "The SQL/XML XML schema"
Note at: At the end of Clause 12, "The SQL/XML XML schema"
Source: WG3:E3A-003
Possible Problem:

Which is normative, Clause 12, "The SQL/XML XML schema", or the actual URL that defines this namespace on-line? If it is the URL, should this clause be downgraded to an informative Annex? If the clause or annex is modified by a TC, is the URL or its contents also changed?

XML-005 The following Possible Problem has been noted:

Severity: Major Technical
Reference: Subclause 7.16, "Mapping SQL data values to XML"
Note at: At the end of Subclause 7.16, "Mapping SQL data values to XML"
Source: WG3:E3A-011
Possible Problem:

The rules for mapping non-null SQL data values to XML do not handle the escaping of the reserved symbols such as <less than operator>, which might be done using either entities (such as **<**) or by escaping the entire string using **CDATA**.

XML-007 The following Possible Problem has been noted:

Severity: Major Technical
Reference: P14, SQL/XML, Subclause 7.3, "Mapping an SQL table to an XML document and an XML schema document"
Note at: GRs of Subclause 7.3, "Mapping an SQL table to an XML document and an XML schema document"
Source: WG3:YYJ-038R1 = H2-2001-373R1
Possible Problem:

Document *XS* is created without declaring a namespace. A user may wish to specify a namespace for this mapping that is used as the value of an xsd:targetNamespace attribute.
Proposed Solution:

None provided with comment.

XML-008 The following Possible Problem has been noted:

Severity: Major Technical
Reference: P14, SQL/XML, Subclause 7.6, "Mapping an SQL table to XML schema data types"
Note at: GRs of Subclause 7.6, "Mapping an SQL table to XML schema data types"
Source: WG3:YYJ-038R1 = H2-2001-373R1 and WG3:YYJ-054 = H2-2001-__
Possible Problem:

Some members feel that the row element is not necessary. Instead of multiple row elements within a table element, they would create multiple elements with the name of the table.

Michael Rys added: Depending on the preferred mapping, a table may be mapped to an XML fragment colleciton and not a document. Thus we would avoid the term XML document and only use the term XML.

Michael also added: We disagree with the proposed mapping and prefer the form (white space only added for formatting purposes):

```
<EMPLOYEE>
  <EMPNO>000010</EMPNO>
  <FIRSTNAME>Christine</FIRSTNAME>
  <LASTNAME>Haas</LASTNAME>
  <BIRTHDATE>1933-08-24</BIRTHDATE>
  <SALARY>52750.00</SALARY>
</EMPLOYEE>
<EMPLOYEE>
  <EMPNO>000020</EMPNO>
  <FIRSTNAME>Michael</FIRSTNAME>
  <LASTNAME>Thompson</LASTNAME>
  <BIRTHDATE>1948-02-02</BIRTHDATE>
  <SALARY>41250.00</SALARY>
<EMPLOYEE>
```

Michael also added: Here are some counter-arguments against the reasons provided in the document:

> First, when we map just the EMPLOYEE table in this way, we then need to create a single root element that contains the sequence of <EMPLOYEE> elements.

There is really no need to provide an element on this level. XML query languages such as XPath or XQuery do not require a single root. Having an implied root for the database or letting the user/tools specify a root in situations where a dump needs to be performed is sufficient.

Thus, there is no need to have this additional level for a default view of a table. A table does not need to correspond to an XML document.

> Our second reason is that a table with no rows in it will not appear at all in the mapping of an SQL schema. A table for which the user does not have SELECT privilege will also not appear at all in this mapping. We believe that and empty table and a table that the user is not allowed to see should have different representations in XML.

This argument forgets to consider that every default XML view of relational data also has an associated schema with it and we could use the schema to disambiguate between the two cases above:

- Table exists, but user has no access: **no entry in the schema**, no data elements.
- Table exists, user has access, no rows: **entry in the schema**, no data elements.

In addition, our product ships with a default XML view mechanism based on not including <row> elements and so far we have not received any feedback from our users or customers that would warrant such an additional level of indirection.
Proposed Solution:

None provided with comment.

XML-009 The following Possible Problem has been noted:

Severity: Major Technical
Reference: P14, SQL/XML, No particular location
Note at: None.
Source: WG3:YYJ-054/H2-2001-_ __

Possible Problem:

We believe it should be left to the implementation on whether the names are fully escaped or partially escaped.
Proposed Solution:

None provided with comment.

XML-010 The following Possible Problem has been noted:

Severity: Major Technical
Reference: P14, SQL/XML, No particular location
Note at: None.
Source: WG3:YYJ-054/H2-2001-_ __
Possible Problem:

There are many alternatives on schema and catalog mappings. Schemata and catalogs in SQL are used to scope table names. In XML, such scoping can be done using either namespaces or local elements as proposed in WG3:YYJ-038R1 = H2-2001-373R1, or in a combination of both.

Thus, if table names are mapped to element names, both are available. Namespaces have the advantage that they provide a fully unique scoping of the names across the different databases (assuming each database server has its own base URI), which provides protecction in information integration scenarios. Namespaces are also the preferred way to associate XML Schema information with the data. Since namespaces should be used anyway for associating XML Schema information with the data, we should consider whether schema and catalog level information is better mapped into the namespace URI.
Proposed Solution:

None provided with comment.

XML-011 The following Possible Problem has been noted:

Severity: Major Technical
Reference: P14, SQL/XML, No particular location
Note at: None.
Source: WG3:YYJ-054/H2-2001-_ __
Possible Problem:

We disagree with the need of introducing artificial names for the table types and suggest anonymous types should be used.

XML elements are a kind of types. Tables as specified in this paper in the relational world are mapped to these elements and do not have a named type in the relational domain. We are of the opinion that this should be preserved in the mapping and lead to anonymous complex types in the generated XML Schema.

This will allow us to map explicitly named table types into named XML complex types in the future without the fear of name clashes. Also the arguments that are made against anonymous types can be countered as follows:

"There is no name by which they can be referenced"

Neither is the structure of a table named except through referring to the table name itself. This is analogous to mapping the structure to an anonymous complex type of the named element. Also, there is no requirement or situation, where this type can be reused anyway.

Typed XML query languages such as XQuery are capable of restricting a typed expression to the element, so even in this context, having a named complex type is not needed.

"and no name by which a tool or application can report information about them. "

Tools and application can report information about the element and its complex type, so there is really no need to create a new named construct.

As mentioned above, introducing such explicit names makes it impossible for the user to use unnamed table types to hide the type names and use explicitly named table types if the type name should be exposed.
Proposed Solution:

None provided with comment.

XML-012 The following Possible Problem has been noted:

Severity: Major Technical
Reference: P14, SQL/XML, No particular location
Note at: None.
Source: WG3:YYJ-054/H2-2001-_ __
Possible Problem:

The preferred way to associate XML schema to instance data is by means of a targetNamespace for the schema and the use of the XML namespace to associate the data. See the examples given below. Using physical locations is bad practice and not scalable over large distributed applications.

Examples using alternate approaches: XML mappings of table EMPLOYEE inside ADMINISTRATOR schema and HR catalog

We would like to illustrate the arguments raised in this paper using the above example with three ways of representing schemas and catalogs: (i) Schema/catalog only in namespace (ii) Only as XML elements or (iii) Both.

However, the simplicity of not having a row element and explicitly named complex types speaks for itself in all three cases.

Schema and catalog only in namespace

The format of the targetnamespace is only an example. Other URI formats could be used. The base URI (http://mydatabase.com) could be either product specific or user specified.

```
<xsd:schema xmlns:xsd="..."
   targetNamespace="http://mydatabase.com?catalog=HR&schema=ADMINISTRATOR">
  <xsd:element name="EMPLOYEE">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="EMPNO">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:length value="6"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="FIRSTNME">
        ...
        </xsd:element>
        ...
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Instance data:

```
<EMPLOYEE
    xmlns="http://mydatabase.com?catalog=HR&schema=ADMINISTRATOR">
  <EMPNO>000010</EMPNO>
  <FIRSTNME>CHRISTINE</FIRSTNME>
  <LASTNAME>HAAS</LASTNAME>
  <BIRTHDATE>1933-08-24</BIRTHDATE>
  <SALARY>52750.00</SALARY>
</EMPLOYEE>
<EMPLOYEE
    xmlns="http://mydatabase.com?catalog=HR&schema=ADMINISTRATOR">
  <EMPNO>000020</EMPNO>
  <FIRSTNME>MICHAEL</FIRSTNME>
  <LASTNAME>THOMPSON</LASTNAME>
  <BIRTHDATE>1948-02-02</BIRTHDATE>
  <SALARY>41250.00</SALARY>
</EMPLOYEE>
```

Schema and catalog only as XML elements

We would still need a namespace to scope against database instances. The problem then is that the Employee Element needs to be local to the schema (and the schema local to the catalog), to avoid conflicts with other employee tables in other catalogs or schemata.

```
<xsd:schema xmlns:xsd="..." targetNamespace="http://mydatabase.com" >
  <xsd:element name="HR">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ADMINISTRATOR">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="EMPLOYEE" minoccurs="0" maxoccurs="unbounded">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="EMPNO">
                      <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                          <xsd:length value="6"/>
                        </xsd:restriction>
                      </xsd:simpleType>
                    </xsd:element>
                    <xsd:element name="FIRSTNME">
                    ...
                    </xsd:element>
                  ...
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Instance data:

```
<HR xmlns="http://mydatabase.com">
  <ADMINISTRATOR>
    <EMPLOYEE>
      <EMPNO>000010</EMPNO>
      <FIRSTNME>CHRISTINE</FIRSTNME>
      <LASTNAME>HAAS</LASTNAME>
      <BIRTHDATE>1933-08-24</BIRTHDATE>
      <SALARY>52750.00</SALARY>
    </EMPLOYEE>
    <EMPLOYEE>
      <EMPNO>000020</EMPNO>
      <FIRSTNME>MICHAEL</FIRSTNME>
      <LASTNAME>THOMPSON</LASTNAME>
      <BIRTHDATE>1948-02-02</BIRTHDATE>
      <SALARY>41250.00</SALARY>
    </EMPLOYEE>
  </ADMINISTRATOR>
</HR>
```

Schema and catalog as XML elements with namespaces

This provides the biggest flexibility at the cost of more XML schemata. It allows to keep the schema and table element declarations global but keeps the schema, catalog and element names in different namespaces. It requires the use of substitution groups.

```
<xsd:schema xmlns:xsd="..." targetNamespace="http://mydatabase.com">
  <!-- database schema, only contains abstract catalog elements -->

  <xsd:element name="catalog" abstract="true"/>
</xsd:schema>

<xsd:schema xmlns:xsd="..." targetNamespace="http://mydatabase.com?catalog=HR"
            xmlns:db="http://mydatabase.com">
  <!-- a specific catalog XML schema, only contains abstract schema elements -->

  <xsd:element name="schema" abstract="true"/>

  <xsd:element name="HR" substitutionGroup="db:catalog">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="schema" minoccurs="0" maxoccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

<xsd:schema xmlns:xsd="..."
            targetNamespace="http://mydatabase.com?catalog=HR&schema=ADMINISTRATOR"
  xmlns:cat="http://mydatabase.com?catalog=HR">
  <!-- a specific relational schema, only contains abstract table elements -->

  <xsd:element name="table" abstract="true"/>

  <xsd:element name="ADMINISTRATOR" substitutionGroup="cat:schema">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="table" minoccurs="0" maxoccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>

<xsd:schema xmlns:xsd="..."
    targetNamespace="http://mydatabase.com/tables?catalog=HR&schema=ADMINISTRATOR"
  xmlns:schema="http://mydatabase.com?catalog=HR&schema=ADMINISTRATOR" >
  <!-- contains the specific tables of a relational schema -->

  <xsd:element name="EMPLOYEE" substitutionGroup="schema:schema">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="EMPNO">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:length value="6"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="FIRSTNME">
        ...
        </xsd:element>
      ...
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Instance data:

```
<HR xmlns="http://mydatabase.com?catalog=HR">
  <ADMINISTRATOR
      xmlns="http://mydatabase.com?catalog=HR&schema=ADMINISTRATOR">
    <EMPLOYEE
        xmlns="http://mydatabase.com/tables?catalog=HR&schema=ADMINISTRATOR">
      <EMPNO>000010</EMPNO>
        <FIRSTNME>CHRISTINE</FIRSTNME>
        <LASTNAME>HAAS</LASTNAME>
        <BIRTHDATE>1933-08-24</BIRTHDATE>
        <SALARY>52750.00</SALARY>
      </EMPLOYEE>
      <EMPLOYEE
          xmlns="http://mydatabase.com/tables?catalog=HR&schema=ADMINISTRATOR">
        <EMPNO>000020</EMPNO>
        <FIRSTNME>MICHAEL</FIRSTNME>
        <LASTNAME>THOMPSON</LASTNAME>
        <BIRTHDATE>1948-02-02</BIRTHDATE>
        <SALARY>41250.00</SALARY>
      </EMPLOYEE>
    </ADMINISTRATOR>
  </HR>
```

Proposed Solution:

None provided with comment.

XML-013 The following Possible Problem has been noted:

Severity: Major Technical
Reference: P14, SQL/XML, Subclause 7.1, "Mapping SQL <identifier>s to XML names"
Note at: None.
Source: Jim Melton
Possible Problem:

SQL/XML currently maps characters of SQL identifiers that cannot be trivially mapped to XML name characters by using an escape sequence. Characters that are not valid for direct use in an XML name whose Unicode position lies on the Basic Multilingual Plane (BMP) are represented by an underscore, a lowercase "x", four hexadecimal digits, and another underscore (_xnnnn_). Characters that are not valid for direct use in an XML name, but whose Unicode position lies outside the BMP are represented by an underscore, a lowercase "x", eight hexadecimal digits, and another underscore. However, the use of eight hexadecimal digits are not required, since The Unicode Standard guarantees that no more than 6 hexadecimal digits will ever be required to represent the Unicode value of any character.

Should Subclause 5.1 be modified to use only 6 hexadecimal digits?
Proposed Solution:

None provided with comment.

XML-014 The following Possible Problem has been noted:

Severity: Major Technical
Reference: P14, SQL/XML, Subclause 9.2, "XQuery syntactic elements"
Note at: Format.
Source: WG3:VIE-018R1 = H2-2002-020R2
Possible Problem:

XQuery rule names and numbers will need to be tracked against changes in XQuery 1.0.
Proposed Solution:

Editor's Notes for WG3:DRS-020 = H2-2002-365

None provided with comment.

Minor Problems and Wordsmithing Candidates:

Index

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