WG3:VIE-011
H2-2002-013
December, 2001

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:VIE-003 = H2-2002-005, *FCD 9075-1 (SQL/Framework)*, December, 2002
2) WG3:VIE-004 = H2-2002-006, *FCD 9075-2 (SQL/Foundation)*, December, 2002
3) WG3:VIE-005 = H2-2002-007, *FCD 9075-3 (SQL/CLI)*, December, 2002
4) WG3:VIE-006 = H2-2002-008, *FCD 9075-4 (SQL/PSM)*, December, 2002
5) WG3:VIE-007 = H2-2002-009, *FCD 9075-9 (SQL/MED)*, December, 2002
6) WG3:VIE-008 = H2-2002-010, *FCD 9075-10 (SQL/OLB)*, December, 2002
7) WG3:VIE-009 = H2-2002-011, *FCD 9075-11 (SQL/Schemata)*, December, 2002
8) WG3:VIE-010 = H2-2002-012, *WD 9075-13 (SQL/JRT)*, December, 2002
9) WG3:VIE-011 = H2-2002-013, *WD 9075-14 (SQL/XML)*, December, 2002

ISO/IEC JTC 1/SC 32

Date: 2001-12-17

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

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 5: Host Language Bindings (SQL/Bindings)

● 1 list element deleted.

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

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

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

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

Annexes A, B, C and D 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, “Mappings”, defines the ways in which certain SQL information can be mapped into XML and certain XML information can be mapped into SQL.

6) Clause 6, “The SQL/XML XML Schema”, defines the content of an XML namespace that is used when SQL and XML are utilized together.

7) Clause 7, “Status codes”, defines values that identify the status of the execution of SQL-statements and the mechanisms by which those values are returned.

8) Clause 8, “Conformance”, specifies the way in which conformance to this part of ISO/IEC 9075 may be claimed.

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

10) 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.

11) 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.

12) Annex D, “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)*.

- About 7 references deleted.

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`

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`

● 1 reference deleted.

(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`

(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.

Insert this paragraph For purposes of this part of ISO/IEC 9075, the definitions given in ISO/IEC 9075-1 and ISO/IEC 9075-2 and the following definitions apply.

3.1.1 Definitions provided in Part 14

to be supplied

3.1.2 Definitions taken from XML

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

a) **NameChar**

b) **Name**

3.2 Notations

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

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>
```

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.

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 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 variable name. The namespace variables and their definitions are shown in Table 1, “Namespace variables and their URIs”.

Table 1—Namespace variables and their URIs

Namespace variable	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:`** as the values of these namespace variables (*i.e.*, to reference these namespaces).

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

4.2 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.

4.2.1 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 **`&lt;`**, **`&amp;`**, **`&gt;`**, **`&apos;`**, and **`&quot;`** 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).

4.2.2 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 5.1, “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.

4.2.3 Mapping SQL data types to XML Schema data types

For each SQL type, there is defined a corresponding XML Schema type. The mapping is fully specified in Subclause 5.15, “Mapping SQL data types to XML Schema data types”. The following is a conceptual description of this mapping.

In general, each SQL predefined type *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.

● 1 paragraph deleted.

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.

4.2.4 Mapping SQL data values to XML data values

For each SQL type *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 5.16, “Mapping SQL data values to XML”. The mappings for values of predefined types are designed to exploit <cast specification> as much as possible.

4.2.5 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*.

4.2.6 Mapping an SQL Table to an XML Document and an XML Schema Document

Subclause 5.3, “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.

4.2.7 Mapping an SQL Schema to an XML Document and an XML Schema Document

Subclause 5.4, “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.

4.2.8 Mapping an SQL Catalog to an XML Document and an XML Schema Document

Subclause 5.5, "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.

4.2.9 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*.

4.2.10 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 5.17, "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 Mappings

5.1 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 3 – 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 T_i be $TM(S_i)$.

5) For each *i* between 1 (one) and *N*, let 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 X_i is implementation-defined.

   b) If S_i is <colon>, then

      Case:

      i) If *i* = 1 (one), then let 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 let X_1 be `_xFFFF_`$T1$.

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 $U_1, U_2, \ldots, U_8$ be the eight <uppercase hexit>s such that T_i is U+$U_1U_2 \ldots U_8$ 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`$U_5U_6U_7U_8$`_`.

NOTE 4 – This case implies that T_i has a UCS-2 encoding, which is U+$U_5U_6U_7U_8$.

2) Otherwise, let X_i be `_x`$U_1U_2U_3U_4U_5U_6U_7U_8$`_`.

NOTE 5 – 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 6 – 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 $X_1, X_2, \ldots$, and X_N in order from left to right.

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

5.2 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 $XMLN_1$ be the XML Name formed by the application of Subclause 5.1, "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.

5.3 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 5.1, "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 5.2, "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 5.10, "Mapping a Collection of SQL Data Types to XML Schema Data Types", to the data types of the columns of *CT*.

   e) Let ***`XST`*** be the result of applying the mapping defined in Subclause 5.6, "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 1, "Namespace variables and their URIs".

   g) Let ***`XSDNS`*** be the value of the namespace definition provided for the namespace variable **`xsd:`** in Table 1, "Namespace variables 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 5.12, "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 1, "Namespace variables 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.

5.4 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 5.1, “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 5.10, “Mapping a Collection of SQL Data Types to XML Schema Data Types”, to the data types of the columns of *CT*.

   d) Let *XMLTYPEN* be the result of applying the mapping defined in Subclause 5.2, “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 5.7, “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 1, “Namespace variables and their URIs”.

   g) Let ***`XSDNS`*** be the value of the namespace definition provided for the namespace variable **`xsd:`** in Table 1, “Namespace variables 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 5.13, “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 1, “Namespace variables 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.

5.5 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 5.1, “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 5.10, “Mapping a Collection of SQL Data Types to XML Schema Data Types”, to the data types of the columns of *CT*.

   d) Let ***XMLTYPEN*** be the result of applying the mapping defined in Subclause 5.2, “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 5.8, “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 1, “Namespace variables and their URIs”.

   g) Let ***XSDNS*** be the value of the namespace definition provided for the namespace variable **`xsd:`** in Table 1, “Namespace variables 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 5.14, "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 1, "Namespace variables 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.

5.6 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 5.1, “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 5.1, “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 5.9, “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 5.1, "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 5.1, "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 5.2, "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 5.2, "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 5.1, "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 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. 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.

5.7 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 5.1, "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 5.1, "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 5.2, "Mapping a multi-part SQL Name to an XML Name", to "TableType", *SC*, *SN*, and *TN*.

   e) Let ***XMLTE***$_i$ be

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

6) Let ***XMLTYPEN*** be the result of applying the mapping defined in Subclause 5.2, "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 5.1, "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.

5.8 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 5.1, "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 5.1, "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 5.2, "Mapping a multi-part SQL Name to an XML Name", to "SchemaType", *SC*, and *SN*.

   e) Let ***`XMLSE`***$_i$ be

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

6) Let ***`XMLTYPEN`*** be the result of applying the mapping defined in Subclause 5.2, "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.

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

5.10 Mapping a Collection of SQL Data Types to XML Schema Data Types

Function

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

General Rules

1) Let n be the number of SQL data types provided for an application of this subclause.

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

3) 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 5.9, "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 5.15, "Mapping SQL data types to XML Schema data types", to D_i.

   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***.

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

5.11 Mapping an SQL Data Type to a Named XML Schema Data Type

Function

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

General Rules

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

2) 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>
```

3) If *D* is a predefined data type that is not a character string data type, then:

   a) Let ***XMLN*** be the result of applying the mapping defined in Subclause 5.9, “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 5.15, “Mapping SQL data types to XML Schema data types”, to *D*.

   c) Case:

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

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

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

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

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

5.12 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 5.1, "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 column sof *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 ***`XMLCN`***$_{\text{j}}$ be the result of applying the mapping defined in Subclause 5.1, "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 ***`XMLC`***$_{\text{j}}$ is the zero-length string.

         2) If V_j is the null value and *NULLS* is nil, then ***`XMLC`***$_{\text{j}}$ is

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

         3) Otherwise:

            A) Let ***`XMLV`***$_{\text{j}}$ be the result of applying the mapping defined in Subclause 5.16, "Mapping SQL data values to XML", to V_j.

            B) ***`XMLC`***$_{\text{j}}$ is

               ```
               <XMLCNj>XMLVj</XMLCNj>
               ```

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.

5.13 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 5.1, "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 ***XMLT***$_i$ be the result of applying the mapping defined in Subclause 5.12, "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.

5.14 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 5.1, “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 ***XMLS***$_i$ be the result of applying the mapping defined in Subclause 5.13, “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.

5.15 Mapping SQL data types to XML Schema data types

Function

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

Syntax Rules

None.

General Rules

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

2) Let *IM* be the mapping of SQL <identifier>s to XML defined in Subclause 5.1, “Mapping SQL <identifier>s to XML Names”.

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

4) Let **`xsd:`** be the XML namespace prefix to be used to identify the XML Schema namespace as shown in Table 1, “Namespace variables and their URIs”.

5) Let **`sqlxml:`** be the XML namespace prefix to be used to identify the XML namespace as shown in Table 1, “Namespace variables 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.

6) 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.

7) 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 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 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>
```

- 1 subrule deleted.

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 7 — *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 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 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 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 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 8 — *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 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:restriction base="xsd:boolean">
    <xsd:annotation>
      <xsd:appinfo>
        <sqlxml:sqltype name="BOOLEAN"/
      </xsd:appinfo>
    </xsd:annotation>
  </xsd:restriction>
</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 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 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 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 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 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 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>
```

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

5.16 Mapping SQL data values to XML

Function

Define the mapping of nonnull SQL data values to XML.

Syntax Rules

None.

General Rules

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

2) Let ***XMLT*** be the XML Schema type obtained by mapping *SQLT* using the Rules of Subclause 5.15, “Mapping SQL data types to XML Schema data types”.

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

4) Let *CV* be the result of

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

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

6) 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*.

● 1 subrule deleted.

   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*.

7) ***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 `&lt;`) or by escaping the entire string using `CDATA`. See Possible Problem **XML-005** in the Editor's Notes.

5.17 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 X_1, X_2 , . . . , X_N be the characters of ***XMLN*** in order from right to left.

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 X_i = '_' (an <underscore>), and X_{i+1} = 'x', and each of X_{i+2}, X_{i+3}, X_{i+4}, and X_{i+5} are all <hexit>s, and X_{i+6} = '_', then

      Case:

      i) If i = 1 (one) and X_3, X_4, X_5, and 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 U_{i+1}, U_{i+2}, U_{i+3}, U_{i+4}, U_{i+5}, and U_{i+6} be the zero-length string.

      iii) Otherwise, U_i, U_{i+1}, U_{i+2}, U_{i+3}, U_{i+4}, U_{i+5}, and U_{i+6} are implementation-defined.

   b) If X_i = '_' (an <underscore>), and X_{i+1} = 'x', and each of X_{i+2}, X_{i+3}, X_{i+4}, X_{i+5}, X_{i+6}, X_{i+7}, X_{i+8}, and X_{i+9}, are all <hexit>s, and X_{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 U_i be the character string of length 1 (one) whose character is UC and let U_{i+1}, U_{i+2}, U_{i+3}, U_{i+4}, U_{i+5}, U_{i+6}, U_{i+7}, U_{i+8}, U_{i+9}, and U_{i+10}, be the zero-length string.

      ii) Otherwise, U_i, U_{i+1}, U_{i+2}, U_{i+3}, U_{i+4}, U_{i+5}, U_{i+6}, U_{i+7}, U_{i+8}, U_{i+9}, and U_{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*"**.

6 The SQL/XML XML Schema

6.1 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/mra/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="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:element name="sqltype">
      <xsd:complexType>
        <xsd:attribute name="name"
          type="sqlxml:typeKeyword"/>
        <xsd:attribute name="length" type="xsd:integer"
          minOccurs="0"/>
        <xsd:attribute name="maxLength"
          type="xsd:integer" minOccurs="0"/>
        <xsd:attribute name="characterSetName"
          type="xsd:string" minOccurs="0"/>
        <xsd:attribute name="collation" type="xsd:string"
          minOccurs="0"/>
        <xsd:attribute name="precision" type="xsd:integer"
          minOccurs="0"/>
        <xsd:attribute name="scale" type="xsd:integer"
          minOccurs="0"/>
        <xsd:attribute name="maxExponent"
          type="xsd:integer" minOccurs="0"/>
        <xsd:attribute name="minExponent"
          type="xsd:integer" minOccurs="0"/>
        <xsd:attribute name="userPrecision"
          type="xsd:integer" minOccurs="0"/>
        <xsd:attribute name="leadingPrecision"
          type="xsd:integer" minOccurs="0"/>
      </xsd:complextType>
    </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 must 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 6, "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.

7 Status codes

This Clause modifies Clause 23, "Status codes", in ISO/IEC 9075-2.

7.1 SQLSTATE

This Subclause modifies Subclause 23.1, "SQLSTATE", in ISO/IEC 9075-2.

Table 2, "SQLSTATE class and subclass values", modifies Table 35, "SQLSTATE class and subclass values", in ISO/IEC 9075-2.

Table 2—SQLSTATE class and subclass values

Category	Condition	Class	Subcondition	Subclass
X	SQL/XML mapping error	0N	(*no subclass*)	000
			unmappable XML Name	001

8 Conformance

This Clause modifies Clause 24, "Conformance", in ISO/IEC 9075-2.

- to be defined -

Annex A
(informative)

SQL Conformance Summary

This Annex modifies Appendix A, "SQL Conformance Summary", in ISO/IEC 9075-2.

- to be supplied -

Annex B
(informative)

Implementation-defined elements

This Annex modifies Appendix 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) Subclause 4.2.1, “Mapping SQL character sets to Unicode”:
    a) The mapping of an SQL character set to Unicode is implementation-defined.
2) Subclause 4.2.9, “Mapping Unicode to SQL character sets”:
    a) The mapping of Unicode to a character set in the SQL-environment is implementation-defined.
3) Subclause 5.1, “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) Subclause 5.17, “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.

Annex C
(informative)

Implementation-dependent elements

This Annex modifies Appendix 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) Subclause 5.15, “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.

● 1 list element.

Annex D
(informative)

SQL feature and package taxonomy

This Annex modifies Appendix 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.

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 4.1, "Namespaces"
Note at: The end of Subclause 4.1, "Namespaces", and in the General Rules of Subclause 5.15, "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 6.1, "The SQL/XML XML Schema"
Note at: At the end of the Syntax Rules of Subclause 6.1, "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 6, "The SQL/XML XML Schema"
Note at: At the end of Clause 6, "The SQL/XML XML Schema"
Source: WG3:E3A-003
Possible Problem:

Which is normative, Clause 6, "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 5.16, "Mapping SQL data values to XML"
Note at: At the end of Subclause 5.16, "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 **`&lt;`**) 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 5.3, "Mapping an SQL Table to an XML Document and an XML Schema Document"
Note at: GRs of Subclause 5.3, "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 5.6, "Mapping an SQL Table to XML Schema Data Types"
Note at: GRs of Subclause 5.6, "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.

Editor’s Notes for WG3:VIE-011 = H2-2002-013

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.