

# ISO/IEC JTC 1/SC 32 N 0634

Date: 2001-04-30

REPLACES: --

<p style="text-align: center;"><b>ISO/IEC JTC 1/SC 32</b></p> <p style="text-align: center;"><b>Data Management and Interchange</b></p> <p style="text-align: center;"><b>Secretariat: United States of America (ANSI)</b></p> <p style="text-align: center;"><b>Administered by Pacific Northwest National Laboratory on behalf of ANSI</b></p>
--

<b>DOCUMENT TYPE</b>	Working Draft Text
<b>TITLE</b>	ISO/IEC WD 9075-14 Information technology - Database languages - SQL - - Part 14: XML-Related Specifications (SQL/XML)
<b>SOURCE</b>	Jim Melton (Editor)
<b>PROJECT NUMBER</b>	1.32.03.05.14.00
<b>STATUS</b>	The document presented here is submitted as a candidate base document followed by two papers containing approved changes to the candidate base document. The Editor has not yet been able to prepare a Working Draft containing the approved changes and presents this document as an informative Working Draft that will be replaced by a proper Working Draft as soon as it has been made available.
<b>REFERENCES</b>	
<b>ACTION ID.</b>	FYI
<b>REQUESTED ACTION</b>	
<b>DUE DATE</b>	
<b>Number of Pages</b>	58
<b>LANGUAGE USED</b>	English
<b>DISTRIBUTION</b>	P & L Members SC Chair WG Conveners and Secretaries

Douglas Mann, Secretariat, ISO/IEC JTC 1/SC 32

Pacific Northwest National Laboratory \*, 901 D Street, SW., Suite 900, Washington, DC, 20024-2115, United States of America

Telephone: +1 703 575 2114; Facsimile: +1 703 671 9180; E-mail: [MannD@battelle.org](mailto:MannD@battelle.org)

available from the JTC 1/SC 32 WebSite <http://www.itc1sc32.org/>

\*Pacific Northwest National Laboratory (PNL) administers the ISO/IEC JTC 1/SC 32 Secretariat on behalf of ANSI

## ISO/IEC JTC 1/SC 32 N00634

# Title: **SQL/XML candidate base document**

Author: Fred Zemke, Ashok Malhotra, Paul Cotton, Michael Rys, Jim Melton  
Source: U.S.A.  
Status: Candidate base document for Part 14, SQL/XML  
Date: February 9, 2001

The following poem is presented solely for the reader's amusement, and is not part of the candidate base document, which begins on the next page.

### Eletelephony

Once there was an elephant  
Who tried to use the telephant -  
No! No! I mean an elephone  
Who tried to use the telephone -  
(Dear me! I am not certain quite  
That even now I've got it right.)

Howe'er it was, he got his trunk  
Entangled in the telephunk;  
The more he tried to get it free,  
The louder buzzed the telephee -  
(I fear I'd better drop the song  
of elehop and telephong!)

Laura E. Richards

The document presented here is submitted as a candidate base document followed by two papers containing approved changes to the candidate base document. The Editor has not yet been able to prepare a Working Draft containing the approved changes and presents this document as an informative Working Draft that will be replaced by a proper Working Draft as soon as it has been made available. Apologies are offered for this delay.

## 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.

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.

International Standard ISO/IEC 9075, Part 14, was prepared by Joint Technical Committee ISO/IEC JTC 1, Information technology.

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)
- Part 7: Temporal (SQL/Temporal)
- 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 sqlxml: namespace”, defines the content of an XML namespace that is used when SQL and XML are utilized together.
- 7) Clause 7, “Conformance”, specifies the way in which conformance to this part of ISO/IEC 9075 may be claimed.
- 8) Annex A, “SQL Conformance Summary”, is an informative Annex. It summarizes the conformance requirements of the SQL language.
- 9) 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.
- 10) 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.
- 11) 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 standards contain provisions that, through reference in this text, constitute provisions of this part of ISO/IEC 9075. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 9075 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

### 2.1 ISO/IEC JTC1 standards

- ISO/IEC 9075-1:1999, *Information technology - Database languages - SQL - Part 1: Framework (SQL/Framework)*.
- ISO/IEC 9075-2:1999, *Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation)*.
- ISO/IEC 9075-3:1999, *Information technology - Database languages - SQL - Part 3: Call-level interface (SQL/CLI)*.
- ISO/IEC 9075-4:1999, *Information technology - Database languages - SQL - Part 4: Persistent stored modules (SQL/PSM)*.
- ISO/IEC 9075-5:1999, *Information technology - Database languages - SQL - Part 5: Host language bindings (SQL/Bindings)*.
- ISO/IEC 10646-1:2000, *Information technology - Universal Multi-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane*.

### 2.2 Publicly-available specifications

- The Unicode Consortium, *The Unicode Standard, Version 3.0*, 2000. ISBN 0-201-61633-5.
- 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>
- (Candidate Recommendation) XML Schema Part 0: Primer, 24 October, 2000, <http://www.w3.org/TR/xmlschema-0/>
- (Candidate Recommendation) XML Schema Part 1: Structure, 24 October, 2000, <http://www.w3.org/TR/xmlschema-1/>
- (Candidate Recommendation) XML Schema Part 2: Datatypes, 24 October, 2000, <http://www.w3.org/TR/xmlschema-2/>

- | - (Candidate Recommendation) Canonical XML Version 1.0, 19 January, 2001,  
<http://www.w3.org/TR/xml-c14n>
- | - (Working Draft) XML Information Set, 2 February, 2001, <http://www.w3.org/TR/xml-infoset>
- (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.

## 3 Definitions, notations and conventions

For purposes of this part of ISO/IEC 9075, the following definitions apply.

### 3.1 Definitions

#### 3.1.1 Definitions taken from XML

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

a) **NameChar**

b) **Name**

#### 3.1.2 Definitions provided in Part 14

- *to be supplied* -

### 3.2 Notations

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 (i.e, not bold), 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.

### 3.3 Conventions

#### 3.3.1 Relationships to other parts of ISO/IEC 9075

##### 3.3.1.1 Clause, Subclause and Table relationships

**Table 1: Clause, Subclause and Table relationships**

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause or Table from another part	Part containing correspondence
Clause 1, "Scope"		

**Table 1: Clause, Subclause and Table relationships**

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause or Table from another part	Part containing correspondence
Clause 2, “Normative references”		
Subclause 2.1, “ISO/IEC JTC1 standards”		
Subclause 2.2, “Publicly-available specifications”		
Clause 3, “Definitions, notations and conventions”		
Subclause 3.1, “Definitions”		
Subclause 3.1.1, “Definitions taken from XML”	<i>(none)</i>	<i>(none)</i>
Subclause 3.1.2, “Definitions provided in Part 14”	<i>(none)</i>	<i>(none)</i>
Subclause 3.2, “Notations”		
Subclause 3.3, “Conventions”		
Subclause 3.3.1, “Relationships to other parts of ISO/IEC 9075”		
Subclause 3.3.1.1, “Clause, Subclause and Table relationships”		
Clause 4, “Concepts”		
Subclause 4.1, “Namespaces”	<i>(none)</i>	<i>(none)</i>
Subclause 4.2, “Mappings”	<i>(none)</i>	<i>(none)</i>
Subcaluse 4.2.1, “Mapping SQL character sets to Unicode”	<i>(none)</i>	<i>(none)</i>
Subclause 4.2.2, “Mapping SQL <identifier>s to XML Names”	<i>(none)</i>	<i>(none)</i>

**Table 1: Clause, Subclause and Table relationships**

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause or Table from another part	Part containing correspondence
Subclause 4.2.3, “Mapping SQL data types to XML Schema data types”	<i>(none)</i>	<i>(none)</i>
Subclause 4.2.4, “Mapping SQL data values to XML data values”	<i>(none)</i>	<i>(none)</i>
Clause 5, “Mappings”	<i>(none)</i>	<i>(none)</i>
Subclause 5.1, “Mapping SQL <identifier>s to XML Names	<i>(none)</i>	<i>(none)</i>
Subclause 5.2, “Mapping SQL data types to XML Schema data types”	<i>(none)</i>	<i>(none)</i>
Clause 6, “The <i>sqlxml:</i> namespace”	<i>(none)</i>	<i>(none)</i>
Clause 7, “Conformance”		
Annex A, “SQL conformance summary”		
Annex B, “Implementation-defined elements”		
Annex C, “Implementation-dependent elements”		
Annex D, “SQL feature and package taxonomy”		

## 4 Concepts

### 4.1 Namespaces

This standard references certain namespaces that are defined by the W3 Consortium or by this standard. Each namespace is referenced using a variable name. The namespace variables and their definitions are shown in Table 2, “Namespace variables and their definitions”

**Table 2: Namespace variables and their definitions**

Namespace variable	Namespace definition
<b>xsd:</b>	http://www.w3.org/2000/10/XMLSchema
<b>xsi:</b>	http://www.w3.org/2000/10/XMLSchema-instance
<b>sqlxml:</b>	<i>to be supplied</i>

A conforming implementation is not required to use the namespace prefixes **xsd:**, **xsi:** or **sqlxml:** as the values of these namespace variables (ie., to reference these namespaces) /

Editor’s Note

The value of the **sqlxml:** namespace must be supplied. 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

The mappings from XML to SQL include:

- *to be defined*

#### 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 5.2, “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, VARCHAR vs. CLOB) 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 CHAR, 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., CHAR when the mapping is not homomorphic, and also VARCHAR and CLOB) the facet **xsd:maxLength** is used. Annotations optionally indicate the precise SQL type (CHAR, VARCHAR or CLOB), the length or maximum length of the SQL type, the character set and the default collation.

The SQL binary string types are mapped to the XML Schema type **xsd:binary**. The latter type has a mandatory facet to select either hex or base 64 encoding. The **sqlxml:** namespace provides the types **sqlxml:binaryhex** and **sqlxml:binarybase64** with the hex and base64 encodings, respectively, already selected. It is implementation-dependent which of these types is used to map the SQL binary types. The **xsd:maxLength** facet is set to the maximum length of the binary string after encoding in XML characters. Annotations optionally indicate the SQL type (BLOB) and the maximum length in octets.

The SQL bit string types are also mapped using either **sqlxml:binaryhex** or **sqlxml:binarybase64**. For a fixed length bit string, the **xsd:length** facet is set; for a varying length bit string, the **xsd:maxLength** facet is set. Note that these facets indicate the length or maximum length of the encoded data in XML characters, not the length of the unencoded data in bits. Annotations optionally indicate the SQL type (BIT or BIT VARYING) and the length or maximum length in bits.

The exact numeric SQL types are mapped to the XML Schema type **xsd:decimal**. In the case of the SQL types NUMERIC and DECIMAL, the **xsd:precision** and **xsd:scale** facets are used, whereas for the SQL types INTEGER and SMALLINT, the **xsd:maxInclusive** and **xsd:minInclusive** facets are used. Annotations optionally indicate the SQL type (NUMERIC, DECIMAL, INTEGER or SMALLINT), 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:timeInstant`. 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:timeDuration`. 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

*to be supplied*

## 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*<sub>1</sub>, *S*<sub>2</sub>, . . . , *S*<sub>*N*</sub> 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 2: 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 and *N*, let  $\mathbf{T}_i$  be *TM*(*S*<sub>*i*</sub>).
- 5) For each *i* between 1 and *N*, let  $\mathbf{x}_i$  be the Unicode character string defined by the following rules.
 

Case:

  - a) If *S*<sub>*i*</sub> has no mapping to Unicode (i.e., *TM*(*S*<sub>*i*</sub>) is undefined), then  $\mathbf{x}_i$  is implementation-defined.

- b) If *S*<sub>*i*</sub> 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”,  $i = 1$ ,  $N \geq 3$ ,  $S_1$  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  $i = 1$  and  $T_1$  is not a valid first character of an XML Name, then:

- i) Let  $U_1, U_2, \dots, U_8$  be the eight <uppercase hexit>s such that  $T_i$  is  $U+U_1U_2\dots U_8$  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 `_xU5U6U7U8_`.

NOTE 3: 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 `_xU1U2U3U4U5U6U7U8_`.

NOTE 4: 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 5: 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  $X_1, X_2, \dots$ , 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 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”
- 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

`http://www.w3.org/2000/10/XMLSchema`

- 5) Let ***sqlxml:*** be the XML namespace prefix to be used to identify the XML namespace

- to be defined -

#### Editor’s Note:

The value of the ***sqlxml:*** namespace identifier must be supplied. See Possible Problem XML-001.

- 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*, respectively, 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>
  <sqlxml:sqltype ANNT ANNL ANNCS ANNCO />
</xsd:annotation>
```

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

```
<xsd:simpleType>
  <xsd:restriction base="xsd:string">
    FACET
    ANN
  </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 *L* be  $2 * N$ . Let **LLIT** be an XML Schema literal denoting *L* in the lexical representation of the XML Schema type **xsd:integer**. Let **EN** be the XML text **binaryhex**.
  - 2) Otherwise, let *L* be the smallest integer greater than  $(8 * N) / 6$ . Let **LLIT** be an XML Schema literal denoting *L* in the lexical representation of the XML Schema type **xsd:integer**. Let **EN** be the XML text **binarybase64**.

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

```
<xsd:maxLength value="LLIT" />
```

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>
  <sqlxml:sqltype ANNT ANNL />
</xsd:annotation>
```

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

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

c) If *SQLT* is a bit 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 bit string in hex or base64. Case:

1) If the encoding is in hex, then let *L* be the smallest integer greater than  $N / 4$ . Let **LLIT** be an XML Schema literal denoting *L* in the lexical representation of the XML Schema type **xsd:integer**. Let **EN** be the XML text **binaryhex**.

2) Otherwise, let *L* be the smallest integer greater than  $N / 6$ . Let **LLIT** be an XML Schema literal denoting *L* in the lexical representation of the XML Schema type **xsd:integer**. Let **EN** be the XML text **binarybase64**.

iii) Case:

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

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

```
<xsd:length value="LLIT" />
```

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

```
name="BIT"
```

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

```
length="NLIT"
```

2) Otherwise:

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

```
<xsd:maxLength value="LLIT" />
```

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

```
name="BIT VARYING"
```

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

```
maxLength="NLIT"
```

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

```
<xsd:annotation>
  <sqlxml:sqltype ANNT ANNL />
</xsd:annotation>
```

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

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

d) 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:precision 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:scale 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 6:  $UP$  may be less than  $P$ , as specified in Syntax Rule 23) of Subclause 6.1 “<data type>” in Part 2 of this international standard.

- 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>
  <sqlxml:sqltype ANNT ANNP ANNS />
</xsd:annotation>
```

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

```
<xsd:simpleType>
```

```

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

```

e) If the type designator of *SQLT* is INTEGER or SMALLINT, 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>
  <sqlxml:sqltype name="INTEGER" />
</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>
  <sqlxml:sqltype name="SMALLINT" />
</xsd:annotation>

```

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

```

<xsd:simpleType>
  <xsd:restriction base="xsd:integer">
    FACETMAX
    FACETMIN
    ANN
  </xsd:restriction>
</xsd:simpleType>

```

f) 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 **ANNNT** 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 **ANNNT** is the zero-length string or

name="DOUBLE PRECISION"

3) Otherwise:

A) It is implementation-dependent whether the XML text **ANNNT** 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 7:  $UP$  may be less than  $P$ , as specified in Syntax Rule 23) of Subclause 6.1 "<data type>" in Part 2 of this international standard.

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>
  <sqlxml:sqltype ANNT ANNP ANNUP ANNMAX ANNMIN />
</xsd:annotation>
```

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

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

- g) 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>
      <sqlxml:sqltype name="BOOLEAN" />
    </xsd:annotation>
  </xsd:restriction>
</xsd:simpleType>
```

- h) 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>
  <sqlxml:sqltype name="DATE" />
</xsd:annotation>
```

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

```
<xsd:simpleType>
```

```

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

```

i) 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>
  <sqlxml:sqltype ANNT ANNS />
</xsd:annotation>

```

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

```

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

```

j) 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

$$(+|-)\backslash\{Nd\}\{2\}:\backslash\{Nd\}\{2\}$$

iii) Case:

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

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

2) Otherwise, let the XML text  $FACETP$  be

```
<xsd:pattern value=
  "\{Nd\}\{2\}:\{Nd\}\{2\}:\{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>
  <sqlxml:sqltype ANNT ANNS />
</xsd:annotation>
```

vii)  $XMLT$  is the XML Schema type defined by

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

k) 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

$$\backslash p \{ Nd \} \{ 4 \} - \backslash p \{ Nd \} \{ 2 \} - \backslash p \{ Nd \} \{ 2 \}$$

iii) Case:

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

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

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

```
<xsd:pattern value=
"DATE\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>
  <sqlxml:sqltype ANNT ANNS />
</xsd:annotation>
```

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

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

1) 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

$$\backslash p \{ Nd \} \{ 4 \} - \backslash p \{ Nd \} \{ 2 \} - \backslash p \{ Nd \} \{ 2 \}$$

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

$$(+|-)\backslash\{Nd\}\{2\}:\backslash\{Nd\}\{2\}$$

iv) Case:

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

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

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

```
<xsd:pattern value=
  "DATE\{Nd\}\{2\}:\{Nd\}\{2\}:\{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>
  <sqlxml:sqltype ANNT ANNS />
</xsd:annotation>
```

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

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

m) 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

$$\backslash p\{Nd\}\{2\}.\backslash p\{Nd\}\{SLIT\}S$$

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

$$\backslash p\{Nd\}\{2\}S$$

iii) Case:

- 1) If *SQLT* is INTERVAL YEAR then:

A) Let the XML text *FACETP* be

$$\langle xsd:pattern\ value="-?P\backslash p\{Nd\}\{PLIT\}Y" / \rangle$$

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

$$\langle xsd:pattern\ value="-?P\backslash p\{Nd\}\{PLIT\}Y\backslash p\{Nd\}\{2\}" / \rangle$$

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

$$\langle xsd:pattern\ value="-?P\backslash p\{Nd\}\{PLIT\}M" / \rangle$$

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}MSECS"/>
```

- 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}MSECS" />
```

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}MSECS" />
```

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="-?PTSECS" />
```

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>  
  <sqlxml:sqltype ANNT ANNP ANNS />  
</xsd:annotation>
```

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

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

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

## 6 The *sqlxml*: namespace

### 6.1 The *sqlxml*: namespace

#### Function

Define the contents of the *sqlxml*: namespace.

#### Syntax Rules

1) The contents of the *sqlxml*: namespace are

```
<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  targetNameSpace="to-be-supplied"
  xmlns:sqlxml="to-be-supplied">
  <xsd:annotation>
    <xsd:documentation>
      ISO/IEC 9075-14:200n SQL/XML
      This document contains definitions of types and
      annotations as specified in ISO/IEC 9075-14:200n.
    </xsd:documentation>
  </xsd:annotation>
```

#### 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

```
<xsd:simpleType name="binaryhex">
  <xsd:restriction base="xsd:binary">
    <xsd:encoding value="hex"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="binarybase64">
  <xsd:restriction base="xsd:binary">
    <xsd:encoding value="base64"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="typeKeyword">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="CHAR"/>
  </xsd:restriction>
</xsd:simpleType>
```

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

<xsd:complexType name="sqltype">
  <xsd:sequence>
    <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:sequence>
</xsd:complexType>

</xsd:schema>
```

## General Rules

None

### Editor's Note

Which is normative, Clause 6, "The sqlxml: namespace" 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.

## 7 Conformance

*- to be defined -*

## **Annex A**

(informative)

### **SQL conformance summary**

*- to be supplied -*

## Annex B

(informative)

### Implementation-defined elements

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 n.n, “Mapping SQL character sets to the XML character set”:
  - a) The mapping of an SQL character set to Unicode is implementation-defined.
- 2) Subcaluse 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.

## Annex C

(informative)

### Implementation-dependent elements

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 5.2, “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.

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

## **Annex D**

(informative)

### **SQL feature and package taxonomy**

*- to be supplied -*

## Possible Problems

[XML-001] The following Possible Problem has been noted:

Severity: Major Technical

Reference: 4.1, “Namespaces”

Note at:

Source: WG3:E3A-003

Possible Problem:

The value of the *sqlxml*: namespace must be supplied.

[XML-002] The following Possible Problem has been noted:

Severity: Major Technical

Reference: Subclause 2.2, “Publicly-available specifications”

Note at:

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. See Possible Problem XML-002.

[XML-003] The following Possible Problem has been noted:

Severity: Major Technical

Reference: Subclause 6.1, “The *sqlxml*: namespace”

Note at: the first <annotation>

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 *sqlxml*: namespace”

Note at: end of the Clause

Source: WG3:E3A-003

Possible Problem:

Which is normative, Clause 6, “The *sqlxml*: namespace” 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?

*- End of paper -*

Title: **Mapping XML Names to SQL  
<identifier>s**

Author: Fred Zemke  
Source: U.S.A.  
Status: Change proposal  
Date: March 18, 2001

## Abstract

This paper proposes a mapping of XML Names to SQL <identifier>s which inverts both the partially escaped and the fully escaped variants of the mapping of SQL <identifier>s to XML Names found in the SQL/XML candidate base document.

## References

- [SQL/XML candidate] Fred Zemke et.al., “SQL/XML candidate base document”, ISO/IEC JTC1/SC32 WG3:E3A-003 = ANSI NCITS H2-2001-009r1
- [SQL/XML overview] Fred Zemke et.al., “Mapping SQL types to XML types - an overview”, ISO/IEC JTC1/SC32 WG3:E3A-005 = ANSI NCITS H2-2001-008r1

## 1. Introduction

The candidate base document for SQL/XML defines a mapping of SQL <identifier>s to XML Names, with two variants, called “partially escaped” and “fully escaped”. As noted in [SQL/XML overview] section 3.2, “Identifier mappings”, these mappings were designed so that a single algorithm can be used to regenerate the SQL <identifier> from the XML Name. This paper proposes that algorithm.

The algorithm has two stages:

1. In the first stage, the XML Name is converted into a possibly different Unicode character string. The algorithm scans the XML Name from left to right. Whenever an escape sequence of the form `_xNNNN_` or `_xNNNNNNNN_` (where *N* is a hexadecimal digit) is found, the escape sequence is replaced by the Unicode character whose code point is `U+NNNN` or `U+NNNNNNNN`.
2. In the second stage, an implementation-defined mapping is used to convert the Unicode character string to SQL\_TEXT. The resulting text is enclosed in double quotes to form a <delimited identifier>.

The following points about the algorithm should be noted:

1. It is possible to have an escape sequence that does not identify the code point of a valid Unicode character. In this case, the conversion is implementation-defined. For example, the implementation may wish to raise an exception condition. Alternatively, the escape sequence might represent, e.g., an ASCII control character such as BEL. The candidate base document already grants the implementation the license to use invalid code points to map such characters to XML Names. Conversely, we must permit the implementation to reverse such implementation-defined mappings.
2. Although the mapping from SQL <identifier>s to XML names only generates uppercase hexadecimal escape sequences, the inverse mapping accepts both uppercase and lowercase hexadecimal escape sequences. The SQLX group felt that it is important for the reverse mapping to be more permissive.
3. In stage 2, it may happen that there is no mapping of the Unicode character string to SQL\_TEXT. In that case the algorithm raises an exception condition: *SQL/XML mapping error - unmappable XML Name*.
4. The result is a <delimited identifier>. If every letter in the result is uppercase, and there is no conflict with a reserved word, then a <regular identifier> might be used. However, the set of reserved words is actually rather fluid, since implementations may define additional reserved words, and the set can evolve over time. I have heard that tool developers typically use only <delimited identifier>s because it immunizes the tool from this issue. Note that every <regular identifier> is equivalent to a <delimited identifier>, so there is no real loss of information.

## 2. Proposal conventions

This proposal uses the following conventions:

- |  |   |
|--|---|
| 1. SMALLCAPS   | denote numbered editorial instructions;   |
| <del>strikeout</del>   | denotes existing text to be deleted;  |
| <b>boldface</b>  | denotes new text to be inserted;  |
| plain  | denotes existing text to be retained,   |
| [ <i>Note:...</i> ]  | brackets enclose italicized notes to the proposal reader  |
| <span style="border: 1px solid black; display: inline-block; width: 40px; height: 15px;"></span> | boxes surround “editing tags,” which are part of the document (not instructions to the editor) and may be deleted, inserted, modified or retained, depending on the typeface within the box |

### 3. Proposal for [XML candidate]

#### 3.1 Changes to 3.3.1.1, “Clause, Subclause and Table relationships”

1. ADD THE FOLLOWING LINES TO TABLE 1, “CLAUSE, SUBCLAUSE AND TABLE RELATIONSHIPS”:

Cause, Subclause or Table in this part of ISO/IEC 9075	Corresponding Clause, Sub-clause of Table from another part	Part containing correspondence
<b>4.2.n, “Mapping Unicode to SQL character sets</b>	<i>(none)</i>	<i>(none)</i>
<b>4.2.n+1, “Mapping XML Names to SQL &lt;identifier&gt;s”</b>	<i>(none)</i>	<i>(none)</i>
<b>5.n, “Mapping XML Names to SQL &lt;identifier&gt;s”</b>	<i>(none)</i>	<i>(none)</i>
<b>N, “Status Codes”</b>		
<b>N.1, SQLSTATE</b>		

#### 3.2 Changes to 4.2, “Mappings

1. EDIT THE LAST TWO PARAGRAPHS AS FOLLOWS:

The mappings from XML to SQL include:

- **mapping Unicode to SQL character sets**
- **mapping XML Names to SQL <identifier>s**
- *more to be defined*

#### 3.3 New subclause 4.2.n and 4.2.n+1

1. ADD THE FOLLOWING AS THE LAST TWO SUBCLAUSES OF 4.2, “MAPPINGS”. NOTE THAT THIS TEXT IS NOT SHOWN IN BOLD, SO THAT THE BOLD MONOSPACE CHARACTERS DENOTING XML TEXT MAY BE VISIBLE.

4.2.n, 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.n+1, 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.n, “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 **\_xNNNN\_** or **\_xNNNNNNNN\_** where **N** denotes a hexadecimal digit. Such sequences are converted to the character of SQL\_TEXT that corresponds to the Unicode code point U+0000NNNN or U+NNNNNNNN, respectively.

NOTE nnn: 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>.

### 3.4 New subclause “Mapping XML Names to SQL <identifier>s”

1. ADD THE FOLLOWING SUBCLAUSE TO CLAUSE 5, “MAPPINGS” (NOTE THAT THE NEW TEXT IS NOT BOLD SO THAT BOLD XML TEXT MAY BE VISIBLE):

5.n 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<sub>1</sub>***, ***X<sub>2</sub>***, ..., ***X<sub>N</sub>*** be the characters of ***XMLN*** in order from right to left.
- 2) Let the *N* Unicode character strings *U<sub>1</sub>*, *U<sub>2</sub>*, . . . , *U<sub>N</sub>* be defined as follows:
 

For *i* from 1 through *N*, in that order:

If *U<sub>i</sub>* has not been determined yet, then

Case:

  - a) If ***X<sub>i</sub>*** = ‘\_’ (underscore); ***X<sub>i+1</sub>*** = ‘x’; ***X<sub>i+2</sub>***, ***X<sub>i+3</sub>***, ***X<sub>i+4</sub>***, and ***X<sub>i+5</sub>***, are all <hexit>s; and ***X<sub>i+6</sub>*** = ‘\_’ (underscore) then

Case:

- i) If  $i = 1$ ; and  $\mathbf{X}_3, \mathbf{X}_4, \mathbf{X}_5$  and  $\mathbf{X}_6$  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  $U+\mathbf{X}_{i+2} \mathbf{X}_{i+3} \mathbf{X}_{i+4} \mathbf{X}_{i+5}$  is a valid Unicode character  $UC$ , then let  $U_i$  be the character string of length 1 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  $\mathbf{X}_i = \text{'\_'} (underscore); \mathbf{X}_{i+1} = \text{'x'}; \mathbf{X}_{i+2}, \mathbf{X}_{i+3}, \mathbf{X}_{i+4}, \mathbf{X}_{i+5}, \mathbf{X}_{i+6}, \mathbf{X}_{i+7}, \mathbf{X}_{i+8}, \mathbf{X}_{i+9}$ , are all <hexit>s; and  $\mathbf{X}_{i+10} = \text{'\_'} (underscore)$  then

Case:

- i) If the Unicode codepoint  $U+\mathbf{X}_{i+2} \mathbf{X}_{i+3} \mathbf{X}_{i+4} \mathbf{X}_{i+5} \mathbf{X}_{i+6} \mathbf{X}_{i+7} \mathbf{X}_{i+8} \mathbf{X}_{i+9}$  is a valid Unicode character  $UC$ , then let  $U_i$  be the character string of length 1 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 whose character is  $\mathbf{X}_i$
- 3) Let  $U$  be the concatenation  $U_1 \parallel U_2 \parallel \dots \parallel U_N$ .
  - 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, 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*".

### 3.5 New Clause N, "Status codes"

1. ADD THE FOLLOWING CLAUSE (BEFORE CLAUSE 7, "CONFORMANCE" LOOKS RIGHT):

#### n Status codes

## n.1 SQLSTATE

Category	Condition	Class	Subcondition	Low class
X	SQL/XML mapping error	<i>(editor's choice)</i>	Unmappable XML Name	<i>(editor's choice)</i>

## 3.6 Changes to Annex B, “Implementation-defined elements”

## 1. EDIT THE ITEMS IN THIS ANNEX AS FOLLOWS:

1) Subclause ~~n.n~~ **4.2.1**, “Mapping SQL character sets to the XML character set”:

a) The mapping of an SQL character set to Unicode is implementation-defined.

**1.1) Subclause 4.2.n, “Mapping Unicode to SQL character sets”**

**a) The mapping of Unicode to a character set in the SQL-environment is implementation-defined.**

2) Subclause **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.

**2.1) Subclause 5.n, “Mapping XML Names to SQL <identifier>s”**

**a) The treatment of an escape sequence of the form `_xNNNN_` or `_xNNNNNNNN_` whose corresponding Unicode code point U+NNNN or U+NNNNNNNN is not a valid Unicode character is implementation-defined.**

## 4. Checklist

Concepts	done
Access Rules	n/a
Conformance Rules	none yet in SQL/XML as a whole
Lists of SQL-statements by category	n/a
Table of identifiers used by diagnostics statements	n/a
Collation coercibility for character strings	n/a
Closing Possible Problems	n/a
Any new Possible Problems clearly identified	none
Reserved and non-reserved keywords	n/a
SQLSTATE tables and Ada package	done
Information and Definition Schemas, including short-name views	n/a
Implementation-defined and –dependent Annexes	done
Incompatibilities Annex	n/a
Embedded SQL and host language implications	n/a
Dynamic SQL issues: including descriptor areas	n/a
CLI issues	n/a

*- End of paper -*

Title: **Mapping nonnull SQL data values to  
XML**

Author: Fred Zemke  
Source: U.S.A.  
Status: Change proposal  
Date: March 18, 2001

## **Abstract**

This paper proposes the mapping of nonnull SQL data values to XML that go with the data type mappings already proposed in the SQL/XML candidate base document.

## **References**

- [Foundation CD] Jim Melton (ed), “Committee Draft (CD) Database Language SQL - Part 2: SQL/Foundation”, ISO/IEC JTC1/SC32 WG3:PER-004 = ANSI NCITS H2-2000-556
- [SQL/XML candidate] Fred Zemke et.al., “SQL/XML candidate base document”, ISO/IEC JTC1/SC32 WG3:E3A-003 = ANSI NCITS H2-2001-009r1
- [SQL/XML overview] Fred Zemke et.al., “Mapping SQL types to XML types - an overview”, ISO/IEC JTC1/SC32 WG3:E3A-005 = ANSI NCITS H2-2001-008r1

## **1. Introduction**

This paper proposes the mapping of nonnull SQL data values to XML that go with the data type mappings already proposed in the SQL/XML candidate base document. The data value mappings are almost totally dictated by the data type mappings. In fact, the only mappings that were not totally dictated were the numeric types. For example, when mapping NUMERIC (p, 0), there was a choice of whether to require a final period, forbid it, or make it implementation-defined. The SQLX group decided that the best thing is to use the definition already found in the SQL <cast specification>. According to [Foundation CD] 6.23 <cast specification> GR 9)a) and 9)b), the cast of a numeric to a character string should result in the shortest literal representing the value. Thus in the case of NUMERIC (p, 0), the decimal point would be forbidden. This also resolves issues of insignificant zeros.

A couple issues regarding data value mappings are not addressed:

1. Mapping of null values was not touched. This is an issue that can only be addressed in a larger context. For elements, there are two possible representations, either by omitting the element or through an attribute that can be set to indicate a null value, whereas for attributes a null value can be indicated only by absence of the attribute.
2. Another issue is how to deal with character strings that contain certain problematic characters such as <less than operator>. There are two possible approaches. One approach uses entities, for example, converting <less than operator> to **&lt;**. The other approach escapes the entire character string using CDATA.

## 2. Proposal conventions

This proposal uses the following conventions:

1. **SMALLCAPS** denote numbered editorial instructions;
- ~~strikeout~~ denotes existing text to be deleted;
- boldface** denotes new text to be inserted;
- plain denotes existing text to be retained,
- [Note:...]* brackets enclose italicized notes to the proposal reader
- boxes surround “editing tags,” which are part of the document (not instructions to the editor) and may be deleted, inserted, modified or retained, depending on the typeface within the box

## 3. Proposal for [XML candidate]

### 3.1 Changes to 3.3.1.1, “Clause, Subclause and Table relationships”

1. ADD THE FOLLOWING LINES TO TABLE 1, “CLAUSE, SUBCLAUSE AND TABLE RELATIONSHIPS”:

Cause, Subclause or Table in this part of ISO/IEC 9075	Corresponding Clause, Sub-clause of Table from another part	Part containing correspondence
<b>5.n, “Mapping SQL data values to XML”</b>	<i>(none)</i>	<i>(none)</i>

### 3.2 Changes to 4.2.4, Mapping SQL data values to XML data values

1. EDIT THIS SUBCLAUSE AS FOLLOWS:

*to be supplied*

**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.n, “Mapping nonnull SQL data**

**values to XML”. The mappings for values of predefined types are designed to exploit <cast specification> as much as possible.**

### 3.3 New Subclause 5.n, “Mapping SQL data values to XML”

1. ADD THE FOLLOWING SUBCLAUSE (FOLLOWING SUBCLAUSE 5.2, “MAPPING SQL DATA TYPES TO XML SCHEMA DATA TYPES” APPEARS APPROPRIATE). NOTE THAT THE NEW TEXT IS NOT SHOWN IN BOLD IN ORDER TO MAKE THE USE OF BOLD TO DENOTE XML TEXT VISIBLE.

5.n Mapping nonnull 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 5.2, “Mapping SQL data types to XML Schema data types”.
- 3) Let *M* be the implementation-defined maximum length of type VARCHAR.
- 4) Let *CV* be the result of  
$$\text{CAST} ( SQLV \text{ AS } \text{VARCHAR} ( M ) )$$
- 5) Let *CSM* be the implementation-defined mapping of the default character set of type VARCHAR 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*.

- c) If *SQLT* is a bit 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*.
- d) If *SQLT* is a numeric type, then let ***XMLV*** be the result of mapping *CV* to Unicode using *CSM*.
- e) 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*.
- f) 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*.
- g) If *SQLT* specifies TIME, then:
- i) Let *P* be the <time fractional seconds precision> of *SQLT*.
  - ii) If *P* is 0, then let *Q* be 0 ; otherwise, let *Q* be *P* + 1.
  - iii) If *SQLT* specifies WITH TIME ZONE, then let *Z* be 6; otherwise let *Z* be 0.
  - 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*.
- h) If *SQLT* specifies TIMESTAMP, then:
- i) Let *P* be the <timestamp fractional seconds precision> of *SQLT*.
  - ii) If *P* is 0, then let *Q* be 0 ; otherwise, let *Q* be *P* + 1.
  - iii) If *SQLT* specifies WITH TIME ZONE, then let *Z* be 6; otherwise let *Z* be 0.
  - 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*.

i) If *SQLT* specifies INTERVAL, then:

i) If *SQLV* is negative, let *SIGN* be '-' (a character string of length 1 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 VARCHAR (*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, then let *Q* be 0; otherwise let *Q* be *P* + 1.

vii) Case:

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

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

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

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

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

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

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

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

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

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

- 12) If *SQLT* is INTERVAL MINUTE TO SECOND, 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, 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-nnn.

### 3.4 New Possible Problem

1. ADD THE FOLLOWING POSSIBLE PROBLEM:

The following Possible Problem has been noted:

Severity: Major Technical

Reference: 5.n, "Mapping SQL data values to XML"

Note at: end of General Rules

Possible Problem:

The rules for mapping nonnull SQL data values to XML do not handle the escaping of the reserved symbols <less than operator>, which might be done using either entities (such as **&lt;**) or by escaping the entire string using **CDATA**.

## 4. Checklist

Concepts	done
Access Rules	n/a
Conformance Rules	none yet in SQL/XML as a whole
Lists of SQL-statements by category	n/a
Table of identifiers used by diagnostics statements	n/a
Collation coercibility for character strings	n/a
Closing Possible Problems	n/a
Any new Possible Problems clearly identified	done
Reserved and non-reserved keywords	n/a
SQLSTATE tables and Ada package	n/a
Information and Definition Schemas, including short-name views	n/a
Implementation-defined and –dependent Annexes	n/a
Incompatibilities Annex	n/a
Embedded SQL and host language implications	n/a
Dynamic SQL issues: including descriptor areas	n/a
CLI issues	n/a

*- End of paper -*