WG3:HBA-010
H2-2003-312
August, 2003

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:HBA-002 = H2-2003-304 = 5WD-01-Framework-2003-09, *WD 9075-1 (SQL/Framework)*, September, 2003
2) WG3:HBA-003 = H2-2003-305 = 5WD-02-Foundation-2003-09, *WD 9075-2 (SQL/Foundation)*, September, 2003
3) WG3:HBA-004 = H2-2003-306 = 5WD-03-CLI-2003-09, *WD 9075-3 (SQL/CLI)*, September, 2003
4) WG3:HBA-005 = H2-2003-307 = 5WD-04-PSM-2003-09, *WD 9075-4 (SQL/PSM)*, September, 2003
5) WG3:HBA-006 = H2-2003-308 = 5WD-09-MED-2003-09, *WD 9075-9 (SQL/MED)*, September, 2003
6) WG3:HBA-007 = H2-2003-309 = 5WD-10-OLB-2003-09, *WD 9075-10 (SQL/OLB)*, September, 2003
7) WG3:HBA-008 = H2-2003-310 = 5WD-11-Schemata-2003-09, *WD 9075-11 (SQL/Schemata)*, September, 2003
8) WG3:HBA-009 = H2-2003-311 = 5WD-13-JRT-2003-09, *WD 9075-13 (SQL/JRT)*, September, 2003

9) WG3:HBA-010 = H2-2003-312 = 5WD-14-XML-2003-09, *WD 9075-14 (SQL/XML)*, September, 2003

ISO/IEC JTC 1/SC 32

Date: 2003-07-25

ISO/IEC 9075-14:2003 (E)

ISO/IEC JTC 1/SC 32/WG 3

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 données — SQL — Partie 14: Specifications à XML (SQL/XML)

Document type: International standard
Document subtype:
Document stage: (4) Approval
Document language: English

Contents

Page

Tables

Table **Page**

Foreword

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

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

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 9075-14 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

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

— Part 1: Framework (SQL/Framework)

— Part 2: Foundation (SQL/Foundation)

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

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

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

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

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

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

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

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

Introduction

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

1) Clause 1, “Scope”, specifies the scope of this part of ISO/IEC 9075.
2) Clause 2, “Normative references”, identifies additional standards that, through reference in this part of ISO/IEC 9075, constitute provisions of this part of ISO/IEC 9075.
3) Clause 3, “Definitions, notations and conventions”, defines the notations and conventions used in this part of ISO/IEC 9075.
4) Clause 4, “Concepts”, presents concepts related to this part of ISO/IEC 9075.
5) Clause 5, “Lexical elements”, defines the lexical elements of the language.
6) Clause 6, “Scalar expressions”, defines the elements of the language that produce scalar values.
7) Clause 7, “Query expressions”, defines the elements of the language that produce rows and tables of data.
8) Clause 8, “Predicates”, defines the predicates of the language.
9) Clause 9, “Mappings”, defines the ways in which certain SQL information can be mapped into XML and certain XML information can be mapped into SQL.
10) Clause 10, “Additional common rules”, specifies the rules for assignments that retrieve data from or store data into SQL-data, and formation rules for set operations.
11) Clause 11, “Additional common elements”, defines additional language elements that are used in various parts of the language.
12) Clause 12, “Schema definition and manipulation”, defines facilities for creating and managing a schema.
13) Clause 13, “SQL-client modules”, defines SQL-client modules and externally-invoked procedures.
14) Clause 14, “Data manipulation”, defines the data manipulation statements.
15) Clause 15, “Control statements”, defines the SQL-control statements.
16) Clause 16, “Session management”, defines the SQL-session management statements.
17) Clause 17, “Dynamic SQL”, defines the SQL dynamic statements.
18) Clause 18, “Embedded SQL”, defines the host language embeddings.
19) Clause 19, “Diagnostics management”, defines the diagnostics management facilities.
20) Clause 20, “Information Schema”, defines viewed tables that contain schema information.
21) Clause 21, “Definition Schema”, defines base tables on which the viewed tables containing schema information depend.
22) Clause 22, “The SQL/XML XML Schema”, defines the content of an XML namespace that is used when SQL and XML are utilized together.

23) Clause 23, “Status codes”, defines values that identify the status of the execution of SQL-statements and the mechanisms by which those values are returned.

24) Clause 24, “Conformance”, specifies the way in which conformance to this part of ISO/IEC 9075 may be claimed.

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

26) 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.

27) 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.

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

29) Annex E, “SQL feature 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 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.

This page intentionally left blank.

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.

This page intentionally left blank.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

2.1 JTC1 standards

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

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

[Schemata] ISO/IEC 9075-11:1999, *Information technology — Database languages — SQL — Part 11: Information and Definition Schemas (SQL/Schemata)*.

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

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

2.2 Other international standards

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

[Unicode 3.1] 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`

[Unicode15] Davis, Mark and Dürst, Martin, *Unicode Standard Annex #15: Unicode Normalization Forms, Version 22.0*, 2002-03-26, The Unicode Consortium
`http://www.unicode.org/unicode/reports/tr15-22.html`

[Unicode19] Davis, Mark, *Unicode Standard Annex #19: UTF-32, Version 9.0*, 2002-03-27, The Unicode Consortium
`http://www.unicode.org/unicode/reports/tr19-9.html`

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

[XPath] *XML Path Language (XPath) Version 1.0*, 16 November, 1999

`http://www.w3.org/TR/xpath`

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

[Schema1] *(Recommendation) XML Schema Part 1: Structures*, 2 May, 2001
`http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/`

[Schema2] *(Recommendation) XML Schema Part 2: Datatypes*, 2 May, 2001
`http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/`

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

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

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

[RFC2396] RFC 2396, *Uniform Resource Identifiers (URS): Generic Syntax*, T. Berners-Lee, R. Fielding, L. Masinter, August, 1998
`http://www.ietf.org/rfc/rfc2396.txt`

[RFC2732] RFC 2732, *Format for Literal IPv6 Addresses in URLs*, R. Hinden, B. Carpenter, L. Masinter, December, 1999
`http://www.ietf.org/rfc/rfc2732.txt`

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.

3.1.1 Definitions taken from XML

For the purposes of this part of ISO/IEC 9075, the definitions of the following terms given in [XML] apply:

3.1.1.1 DTD

3.1.1.2 well-formed

3.1.1.3 well-formedness constraint

3.1.1.4 XML declaration

3.1.1.5 XML document

3.1.2 Definitions taken from XML Schema

For the purposes of this part of ISO/IEC 9075, the definitions of the following terms given in [Schema1] and [Schema2] apply:

3.1.2.1 annotation

3.1.2.2 facet

3.1.2.3 value space

3.1.3 Definitions provided in Part 14

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

3.1.3.1 all group content model: The content model of an XML Schema complex type described with **`xs:all`** as defined in [Schema1].

3.1.3.2 **canonical XML Schema literal:** A canonical lexical representation for an XML Schema type ***XST***, as defined in [Schema2]. There is a unique canonical XML Schema literal for each value in the value space of ***XST***.

3.1.3.3 **metadata:** Data about data. In this International Standard, metadata is included in table descriptors, column descriptors, and so forth, as defined in ISO/IEC 9075-2 and other parts of this International Standard.

3.1.3.4 **sequence content model:** The content model of an XML Schema complex type described with `xs:sequence` as defined in [Schema1].

3.1.3.5 **URI:** Uniform Resource Identifier as defined in [RFC2396], as updated by [RFC2732].

3.1.3.6 **valid XML character:** A legal character as defined in [XML].

3.1.3.7 **XML attribute:** An attribute as defined by [XML].

3.1.3.8 **XML attribute information item:** An attribute information item, as defined in [Infoset].

3.1.3.9 **XML character information item:** A character information item, as defined in [Infoset].

3.1.3.10 **XML comment information item:** Aa comment information item, as defined in [Infoset].

3.1.3.11 **XML document information item:** A document information item, as defined in [Infoset].

3.1.3.12 **XML document type declaration information item:** A document type declaration information item, as defined in [Infoset].

3.1.3.13 **XML element:** An element as defined by [XML].

3.1.3.14 **XML element information item:** An element information item, as defined in [Infoset].

3.1.3.15 **XML information item:** An information item, as defined in [Infoset].

3.1.3.16 **XML Name:** A Name as defined by [XML].

3.1.3.17 **XML NameChar:** A NameChar as defined by [XML].

3.1.3.18 **XML namespace:** An XML namespace as defined by [Namespaces].

3.1.3.19 **XML namespace prefix:** A namespace prefix as defined by [Namespaces].

3.1.3.20 **XML namespace information item:** A namespace information item, as defined in [Infoset].

3.1.3.21 **XML NCName:** An NCName, as defined by rule [4] in [Namespaces].

3.1.3.22 **XML notation information item:** A notation information item, as defined in [Infoset].

3.1.3.23 **XML processing instruction information item:** A processing instruction information item, as defined in [Infoset].

3.1.3.24 **XML QName:** A QName, as defined by rule [6] of [Namespaces].

3.1.3.25 **XML QName prefix:** A Prefix, as defined by rule [7] of [Namespaces].

3.1.3.26 **XML QName local part:** A LocalPart, as defined by rule [8] of [Namespaces].

3.1.3.27 **XML Schema:** A Schema (or schema) as defined by [Schema1].

3.1.3.28 **XML Schema built-in data type:** A built-in datatype, as defined in [Schema2].

3.1.3.29 **XML Schema complex type:** A complex type defined by a complex type definition, as defined in [Schema1].

3.1.3.30 **XML Schema data type:** A datatype, as defined in [Schema2].

3.1.3.31 **XML Schema document:** A schema document, as defined in [Schema1].

3.1.3.32 **XML Schema simple type:** A simple type defined by a simple type definition, as defined in [Schema2].

3.1.3.33 **XML Schema type:** A term used to collectively refer to XML Schema built-in data types, XML Schema complex types, XML Schema data types, and XML Schema simple types, when it is not necessary to distinguish among those terms.

3.1.3.34 **XML target namespace:** A target namespace as defined by [Schema1].

3.1.3.35 **XML target namespace URI:** The URI of an XML target namespace.

3.1.3.36 **XML text:** A character string that is a substring of a textual XML content, as defined in Subclause 10.14, “Parsing a character string as an XML value”.

3.1.3.37 **XML unexpanded entity reference information item:** An unexpanded entity reference information item, as defined in [Infoset].

3.1.3.38 **XML unparsed entity information item:** An unparsed entity information item, as defined in [Infoset].

3.2 Notation

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

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

```
<xsd:element>
```

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

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

This page intentionally left blank.

4 Concepts

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

4.1 Data types

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

4.1.1 Naming of predefined types

This Subclause modifies Subclause 4.1.2, "Naming of predefined types", in ISO/IEC 9075-2.

Insert after the 1st sentence of the 1st paragraph SQL defines a predefined data type named by the following <key word>: XML.

Augment the 2nd paragraph

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

4.1.2 Comparison and ordering

This Subclause modifies Subclause 4.1.4, "Comparison and ordering", in ISO/IEC 9075-2.

Augment the 4th paragraph

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

4.2 XML

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

— The name of the data type (XML).

4.2.1 Characteristics of XML values

An *XML root information item* is a data item defined in the manner of [Infoset] as follows:

An XML root information item has the following properties:

— [children] An ordered list of child information items. The list may contain zero or one XML document type declaration information item, zero or more XML element information item, zero or more XML character information item, zero or more processing instruction information items, and zero or more comment information items. The XML document type declaration information item, if present, is the first information item in the list. There are no constraints on the order of other child information items; in particular, child information items of the various sorts may be intermingled.

— [notations] An unordered set of XML notation information items. Support for this property is implementation-defined.

— [unparsed entities] An unordered set of XML unparsed entity information items. Support for this property is implementation-defined.

— [standalone] An indication of the standalone status of the XML value, either “yes” or “no” or “no value”.

— [version] A string representing the XML version of the XML value represented by the XML root information item, or the special value “no value”.

NOTE 1 — An XML root information item is similar to an XML document information item, with the following modifications: an XML root information item does not have a [document element] property, a [base URI] property, a [character encoding scheme] property, or an [all declarations processed] property; and the [children] property permits more than one XML element information item.

An *SQL/XML information item* is either an XML root information item or one of the following (defined in Subclause 3.1.3, “Definitions provided in Part 14”): an XML attribute information item, an XML character information item, an XML comment information item, an XML document type declaration information item, an XML element information item, an XML namespace information item, an XML notation information item, an XML processing instruction information item, an XML unexpanded entity reference information item, or an XML unparsed entity information item.

An *XML value* is either the null value, or a collection of SQL/XML information items, consisting of exactly one XML root information item, plus any other SQL/XML information items that can be reached recursively by traversing the properties of the SQL/XML information items.

NOTE 2 — A future version of this part of ISO/IEC 9075 may use a data model based on the work of the XML Query Working Group of the World Wide Web Consortium.

4.2.2 XML comparison and assignment

Values of XML type are assignable to sites of XML type.

XML values are not comparable.

4.2.3 Operations involving XML values

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

<XML forest> is an operator that returns an XML value given a list of <forest element>s. An XML element is produced from each <forest element>, using the column name or, if provided, the <forest element name> as the XML element name and the <forest element value> as the element content. The value of <forest element value> can be any value that has a mapping to an XML value.

<XML concatenation> is an operator that returns an XML value by concatenating a list of XML values, as defined in Subclause 6.9, “<XML concatenation>”.

<XML comment> is an operator that returns an XML value, given a character string *CS*. The XML value consists of an XML root information item with one child, an XML comment information item whose [content] property is *CS*, mapped to Unicode.

<XML PI> is an operator that returns an XML value, given an <identifier> and an optional character string *CS*. The XML value consists of an XML root information item with one child, an XML processing information item whose [target] property is the partially escaped mapping of the <identifier> to an XML Name, and whose [content] property is *CS*, trimmed of leading blanks and mapped to Unicode.

4.3 Data analysis operations (involving tables)

This Subclause modifies Subclause 4.15, “Data analysis operations (involving tables)”, in ISO/IEC 9075-2.

4.3.1 Aggregate functions

This Subclause modifies Subclause 4.15.4, “Aggregate functions”, in ISO/IEC 9075-2.

Add to the bulleted list following the 7th paragraph

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

4.4 SQL-invoked routines

This Subclause modifies Subclause 4.27, “SQL-invoked routines”, in ISO/IEC 9075-2.

4.4.1 Routine descriptors

This Subclause modifies Subclause 4.27.4, “Routine descriptors”, in ISO/IEC 9075-2.

Augment the routine descriptor of external routines

— For every SQL parameter that has an associated string type, the routine descriptor includes the character string type descriptor of the associated string type.

— For every SQL parameter that has an associated XML option, the routine descriptor includes an indication of the associated XML option.

4.5 SQL-statements

This Subclause modifies Subclause 4.33, “SQL-statements”, in ISO/IEC 9075-2.

4.5.1 SQL-statements classified by function

This Subclause modifies Subclause 4.33.2, “SQL-statements classified by function”, in ISO/IEC 9075-2.

4.5.1.1 SQL-session statements

This Subclause modifies Subclause 4.33.2.7, “SQL-session statements”, in ISO/IEC 9075-2.

$\boxed{\text{Insert this paragraph}}$ The following are additional SQL-session statements:

— <set XML option statement>

4.6 SQL-sessions

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

4.6.1 SQL-session properties

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

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

$\boxed{\text{Insert at the end of dash list in 12nd paragraph}}$

— The current XML option.

4.7 XML namespaces

This standard references certain XML namespaces that are defined by the World-Wide Web Consortium or by this standard. Each XML namespace is referenced using an XML namespace prefix. The XML namespace prefixes and their definitions are shown in Table 1, “XML namespace prefixes and their URIs”.

Table 1 — XML namespace prefixes and their URIs

XML namespace prefix	XML namespace URI
`xsd`	`http://www.w3.org/2001/XMLSchema`
`xsi`	`http://www.w3.org/2001/XMLSchema-instance`
`xq`	`http://www.w3.org/2002/11/xquery-functions`
`sqlxml`	`http://standards.iso.org/iso/9075/2003/sqlxml`

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

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

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

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

It is intended that the contents of that file be identical to the contents of Clause 22, “The SQL/XML XML Schema”. This file has been created for the convenience of the implementors of this specification.

4.8 Overview of mappings

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

— Mapping SQL character sets to Unicode.

— Mapping SQL <identifier>s to XML Names.

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

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

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

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

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

The mappings from XML to SQL include:

— Mapping Unicode to SQL character sets.

— Mapping XML Names to SQL <identifier>s.

4.8.1 Mapping SQL character sets to Unicode

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

NOTE 3 — The entity references **`&lt;`**, **`&amp;`**, **`&gt;`**, **`&apos;`**, and **`&quot;`**, as defined by [XML] are regarded as each representing a single character in XML, and do not pose an obstacle to defining homomorphic mappings.

4.8.2 Mapping SQL <identifier>s to XML

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

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

4.8.3 Mapping SQL data types to XML

For each SQL type or domain, with the exception of structured types, reference types, and the interval type, there is a corresponding XML Schema type. The mapping is fully specified in Subclause 9.15, "Mapping SQL data types to XML Schema data types". The following is a conceptual description of this mapping.

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

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

The SQL character string types are mapped to the XML Schema type **`xsd:string`**. For the SQL type CHARACTER, if the mapping of the SQL character set to Unicode is homomorphic, then fixed length strings are mapped to fixed length strings, and the facet **`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`**. 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. For <XML element> and <XML forest>, the choice of whether to map to **`xsd:hexBinary`** or **`xsd:base64Binary`** is governed by the innermost <XML binary encoding> whose scope includes the <XML element> or <XML forest>; the default is implementation-defined. When mapping an SQL table, schema or catalog to XML, the choice is governed by a parameter, as specified in Subclause 9.3, "Mapping an SQL table to XML and an XML Schema document", Subclause 9.4, "Mapping an SQL schema to an XML document and an XML Schema document", and Subclause 9.5, "Mapping an SQL catalog to an XML document and an XML Schema document".

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

The approximate numeric SQL types are mapped to either the XML Schema type **`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 displacement. Optionally, an annotation indicates the SQL type, DATE.

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

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

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

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

An SQL distinct type is mapped to an XML Schema simple type by mapping the source type of the distinct type.

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

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

4.8.4 Mapping values of SQL data types to XML

For each SQL type or domain *SQLT*, there is also a mapping of values of type *SQLT* to the value space of the corresponding XML Schema type. The mappings of values are largely determined by the data type mappings. The precise rules for non-null values are found in Subclause 9.16, "Mapping values of SQL data types to values of XML Schema data types". The mappings for values of predefined types are designed to exploit <cast specification> as much as possible. As for null values, there is generally a choice of whether to represent nulls using absence or **`xsi:nil="true"`**. However, for elements of a collection type, null values are always represented by **`xsi:nil="true"`**.

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

An *XML unmappable data type* is a data type that is one of the following: a structured type, a reference type, and the interval type. An *XML unmappable column* is a column that has a declared type that is one of the XML unmappable data types or has a declared type that is based on an XML unmappable data type.

A column *C* of table *T* is a *visible column* of *T* for authorization identifier *U* if the applicable privileges for *U* include the SELECT privilege on *C* and, if the declared type of *C* is a distinct type, the applicable privileges for *U* include EXECUTE on the user-defined cast function identified by the Syntax Rules of Subclause 6.12, "<cast specification>", in ISO/IEC 9075-2. A column *C* of table *T* is an *XML visible column* of *T* for authorization identifier *U* if the declared type of *C* is not an XML unmappable data type.

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

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

4.8.6 Mapping an SQL table to XML

Subclause 9.3, "Mapping an SQL table to XML and an XML Schema document", defines a mapping of an SQL table to an XML Schema document that describes the structure of the mapped XML and either an XML document or an XML forest of elements. Only base tables and viewed tables may be the source of this mapping.

NOTE 4 — This part of this International Standard specifies no syntax for invoking the mapping specified in Subclause 9.3, "Mapping an SQL table to XML and an XML Schema document". This specification is intended to be used by applications and referenced by other standards.

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

This mapping allows the invoker to specify:

— Whether to map the table to an XML forest of elements where the name of each top-level element is derived from the table name and represents a row in the table, or to map the table to an XML document with a single root element whose name is derived from the table name and to map each row to an element named **`<row>`**.

— The XML target namespace URI of the XML Schema and data to be mapped (if the XML target namespace URI is specified as a zero-length string, then no XML namespace is added).

— Whether to map null values to absent elements, or whether to map them to elements that are marked with **`xsi:nil="true"`**.

Some of the XML Schema type definitions and element definitions may contain annotations to represent SQL metadata that is not directly relevant to XML. It is implementation-defined whether these annotations are generated.

4.8.7 Mapping an SQL schema to XML

Subclause 9.4, “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 represents the data in these tables, and an XML Schema document that describes the first document.

NOTE 5 — This part of this International Standard specifies no syntax for invoking the mapping specified in Subclause 9.4, “Mapping an SQL schema to an XML document and an XML Schema document”. This specification is intended to be used by applications and referenced by other standards.

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

This allows the invoker to specify:

— Whether to map the table to an XML forest of elements where the name of each top-level element is derived from the table name and represents a row in the table, or to map the table to an XML document with a single root element whose name is derived from the table name and to map each row to an element named **`<row>`**.

— The XML target namespace URI of the XML Schema and data to be mapped (if the XML target namespace URI is specified as a zero-length string, then no XML namespace is added).

— Whether to map null values to absent elements, or whether to map them to elements that are marked with **`xsi:nil="true"`**.

Some of the XML Schema type definitions and element definitions may contain annotations to represent SQL metadata that is not directly relevant to XML. It is implementation-defined whether these annotations are generated.

The SQL schema mapping assumes an implementation-dependent *repeatable ordering* when iterating over the XML visible tables in the SQL schema. This allows generating the correct alignment of the table data with the element declarations in situations where the generated XML Schema uses a sequence content model instead of an all group content model.

4.8.8 Mapping an SQL catalog to XML

Subclause 9.5, “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 represents the data in these tables, and an XML Schema document that describes the first document.

NOTE 6 — This part of this International Standard specifies no syntax for invoking the mapping specified in Subclause 9.5, “Mapping an SQL catalog to an XML document and an XML Schema document”. This specification is intended to be used by applications and referenced by other standards.

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

This mapping allows the user that invokes this mapping to specify:

— Whether to map the table to an XML forest of elements where the name of each top-level element is derived from the table name and represents a row in the table, or to map the table to an XML document with a single root element whose name is derived from the table name and each row is mapped to an element names **`<row>`**.

— The XML target namespace URI of the XML Schema and data to be mapped (if the XML target namespace URI is specified as a zero-length string, then no XML namespace is added).

— Whether to map null values to absent elements, or whether to map them to elements that are marked with **`xsi:nil="true"`**.

Some of the XML Schema type definitions and element definitions may contain annotations to represent SQL metadata that is not directly relevant to XML. It is implementation-defined whether these annotations are generated.

4.8.9 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*.

4.8.10 Mapping XML Names to SQL

A single algorithm suffices to reverse both the partially escaped and the fully escaped variants of the mapping of SQL <identifier>s to XML Names. This algorithm is found in Subclause 9.17, “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`*`NNNNNN`*`_`** where ***`N`*** denotes a hexadecimal digit. Such sequences are converted to the character of SQL_TEXT that corresponds to the Unicode code point U+0000*NNNN* or U+00*NNNNNN*, respectively.

NOTE 7 — This part of this International Standard specifies no syntax for invoking the mapping specified in Subclause 9.17, “Mapping XML Names to SQL <identifier>s”. This specification is intended to be used by applications and referenced by other standards.

NOTE 8 — 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>.

This page intentionally left blank.

5 Lexical elements

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

5.1 <token> and <separator>

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

Function

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

Format

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

  | BASE64

  | CONTENT

  | DOCUMENT

  | HEX

  | PRESERVE

  | STANDALONE | STRIP

  | VERSION

  | WHITESPACE

<reserved word> ::=

    !! All alternatives from ISO/IEC 9075-2

  | XML | XMLAGG | XMLATTRIBUTES | XMLBINARY | XMLCONCAT
  | XMLCOMMENT | XMLELEMENT | XMLFOREST
  | XMLNAMESPACES | XMLPARSE | XMLPI | XMLROOT | XMLSERIALIZE
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

6 Scalar expressions

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

6.1 <data type>

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

Function

Specify a data type.

Format

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

<XML type> ::= XML
```

Syntax Rules

1) Insert after SR 40) XML specifies the XML data type.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 20) If <data type> is an <XML type>, then an XML type descriptor is created, including the name of the data type (XML).

Conformance Rules

1) Insert this CR Without Feature X010, “XML type”, conforming SQL language shall not contain an <XML type>.

2) $\boxed{\text{Insert this CR}}$ Without Feature X011, “Arrays of XML type”, conforming SQL language shall not contain an <array type> that is based on a <data type> that is either the XML type or a distinct type whose source type is the XML type.

3) $\boxed{\text{Insert this CR}}$ Without Feature X012, “Multisets of XML type”, conforming SQL language shall not contain a <multiset type> that is based on a <data type> that is either the XML type or a distinct type whose source type is the XML type.

6.2 <field definition>

Function

Define a field of a row type.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) $\boxed{\text{Insert this CR}}$ Without Feature X015, “Fields of XML type”, conforming SQL language shall not contain a <field definition> that contains a <data type> that is based on either the XML type or a distinct type whose source type is the XML type.

6.3 <cast specification>

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

Function

Specify a data conversion.

Format

No additional Format items.

Syntax Rules

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

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

Where:

EN	= Exact Numeric
AN	= Approximate Numeric
C	= Character (Fixed- or Variable-length, or character large object)
FC	= Fixed-length Character
VC	= Variable-length Character
CL	= Character Large Object
D	= Date
T	= Time
TS	= Timestamp

```
  YM  = Year-Month Interval
  DT  = Day-Time Interval
  BO  = Boolean
 UDT  = User-Defined Type
  BL  = Binary Large Object
  RT  = Reference type
  CT  = Collection type
  RW  = Row type
  DL  = Datalink
 XML  = XML type
```

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 2) If *SD* is XML, then *TV* is *SV*.

Conformance Rules

1) Insert this CR Without Feature X010, “XML type”, conforming SQL language shall not contain a <cast operand> whose declared type is XML.

6.4 <value expression>

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

Function

Specify a value.

Format

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

Syntax Rules

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

2) Insert after SR 7)n) An <aggregate function> that specifies <XML aggregate>.

Access Rules

No additional Access Rules.

General Rules

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

Conformance Rules

No additional Conformance Rules.

6.5 <string value function>

This Subclause modifies Subclause 6.29, "<string value function>", in ISO/IEC 9075-2.

Function

Specify a function yielding a value of type character string or binary string.

Format

```
<character value function> ::=
    !! All alternatives for ISO/IEC 9075-2
  | <XML serialize>

<XML serialize> ::=
    XMLSERIALIZE <left paren> <document or content> <XML value expression>
    AS <data type> <right paren>

<document or content> ::=
    DOCUMENT
  | CONTENT
```

Syntax Rules

1) Replace SR 2) The declared type of <character value function> is the declared type of the immediately contained <character substring function>, <regular expression substring function>, <fold>, <transcoding>, <character transliteration>, <trim function>, <character overlay function>, <normalize function>, <specific type method>, or <XML serialize>.

2) Insert this SR If <XML serialize> is specified, then <data type> shall be a character string type. The declared type of <XML serialize> is <data type>.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 2) The result of <character value function> is the result of the immediately contained <character substring function>, <regular expression substring function>, <fold>, <transcoding>, <character transliteration>, <trim function>, <character overlay function>, <normalize function>, <specific type method>, or <XML serialize>.

2) Insert this GR If <XML serialize> is specified, then let *DC* be the <document or content>, let *XMLV* be the value of the <XML value expression>, and let *DT* be the <data type>. The result of <XML serialize> is the result determined by applying the General Rules of Subclause 10.13, "Serialization of an XML value", with *DC* as *SYNTAX*, *XMLV* as *VALUE*, and *DT* as *TYPE*.

Conformance Rules

1) Without Feature X070, “XMLSerialize: CONTENT option”, in conforming SQL language, <XML serialize> shall not immediately contain a <document or content> that is CONTENT.

2) Without Feature X071, “XMLSerialize: DOCUMENT option”, in conforming SQL language, <XML serialize> shall not immediately contain a <document or content> that is DOCUMENT.

6.6 <XML value expression>

Function

Specify an XML value.

Format

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

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

Syntax Rules

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

Access Rules

None.

General Rules

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

Conformance Rules

1) Without Feature X010, “XML type”, conforming SQL language shall not contain an <XML value expression>.

6.7 <XML value function>

Function

Specify a function that yields a value of type XML.

Format

```
<XML value function> ::=
    <XML comment>
  | <XML concatenation>
  | <XML element>
  | <XML forest>
  | <XML parse>
  | <XML PI>
  | <XML root>
```

Syntax Rules

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

Access Rules

None.

General Rules

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

Conformance Rules

1) Without Feature X010, “XML type”, conforming SQL language shall not contain an <XML value function>.

6.8 <XML comment>

Function

Generate an XML comment.

Format

```
<XML comment> ::=
    XMLCOMMENT <left paren> <string value expression> <right paren>
```

Syntax Rules

1) The declared type of <XML comment> is XML.

Access Rules

None.

General Rules

1) Case:

   a) If If the value of the <string value expression> is the null value, then the result is the null value.

   b) Otherwise, let SVE be the <string value expression>.

      i) If the value of

         ```
         '<!--' || SVE || '-->'
         ```

         does not conform to rule [15], “comment”, of [XML], then an exception condition is raised: *data exception — invalid comment*.

      ii) Let *XCII* be an XML comment information item with the following properties:

         1) The value of the [content] property is the value of *SVE*, converted to Unicode using the implementation-defined mapping from the character set of *SVE* to Unicode.

         2) The value of the [parent] information item is initially set to 'unknown'.

      iii) Let *XRII* be an XML root information item with the following properties:

         1) The [children] property is a list with one element, *XCII*.

         2) The [notations] property is the empty set.

         3) The [unparsed entities] property is the empty set.

         4) The [version] property is 'no value'.

5) The [standalone] property is 'no value'.

iv) The [parent] property of *XCII* is set to *XRII*.

v) The result of the <XML comment> is the XML value consisting of *XRII* and *XCII*.

Conformance Rules

1) Without Feature X036, “XMLComment”, conforming SQL language shall not contain an <XML comment>.

6.9 <XML concatenation>

Function

Concatenate a list of XML values.

Format

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

Syntax Rules

None.

Access Rules

None.

General Rules

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

2) Let R_0 be the null value.

3) For all i, 1 (one) $\leq i \leq n$, let R_i be the concatenation of R_{i-1} and XV_i, as determined by applying the General Rules of Subclause 10.12, “Concatenation of two XML values”.

4) The result of <XML concatenation> is R_n.

Conformance Rules

1) Without Feature X020, “XML concatenation”, conforming SQL language shall not contain an <XML concatenation>.

6.10 <XML element>

Function

Generate an XML value with a single XML element information item as a child of its XML root information item.

Format

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

<XML element name> ::= <identifier>

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

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

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

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

<XML attribute name> ::= <identifier>

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

Syntax Rules

1) The scope of the <XML namespace declaration> is the <XML element>.

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

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

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

   b) Let AV_i be the <XML attribute value> of A_i.

   c) The declared type of AV_i shall be a distinct type or a predefined type other than XML.

   d) Case:

      i) If A_i contains an <XML attribute name> AN_i, then let ANC_i be the result of the partially escaped mapping from an SQL <identifier> to an XML Name specified in Subclause 9.1, "Mapping SQL <identifier>s to XML Names", applied to AN_i.

      ii) Otherwise, AV_i shall be a <column reference>. Let CN_i be the <column name> of the column designated by the <column reference> that is AV_i. Let ANC_i be the result of the fully escaped mapping from an SQL <identifier> to an XML Name specified in Subclause 9.1, “Mapping SQL <identifier>s to XML Names”, applied to CN_i.

e) ANC_i shall be an XML QName.

f) ANC_i shall not be equivalent to '**`xmlns`**', and ANC_i shall not have an XML QName prefix that is equivalent to '**`xmlns`**'.

g) The Syntax Rules of Subclause 10.8, “Determination of [namespace name] property”, are applied, with ANC_i as *QNAME* and with the <XML element> as *BNFTERM*, resulting in an XML namespace URI $NSURI_i$.

4) There shall not be two <XML attribute>s A_i and A_j such that *i* does not equal *j*, $NSURI_i$ equals $NSURI_j$ according to the UCS_BASIC collation, and the XML QName local part of ANC_i equals the XML QName local part of ANC_j.

5) Let *EN* be the character representation of <XML element name> after the partially escaped mapping from an SQL <identifier> to an XML Name specified in Subclause 9.1, “Mapping SQL <identifier>s to XML Names”, has been applied. *EN* shall be an XML QName.

6) The Syntax Rules of Subclause 10.8, “Determination of [namespace name] property”, are applied, with *EN* as *QNAME* and with the <XML element> as *BNFTERM*, resulting in an XML namespace URI *ENSURI*.

7) For each <XML element content> *XEC*, the declared type of *XEC* shall be a predefined type, a distinct type, or a collection type.

Access Rules

1) If the declared type *DT* of the <value expression> *VE* immediately contained in an <XML attribute value> or an <XML element content> is a distinct type, then let *ST* be the source type of *DT*. The Access Rules of Subclause 6.12, “<cast specification>”, in ISO/IEC 9075-2, are applied to:

```
CAST ( VE AS ST )
```

General Rules

1) Let *EI* be an element information item whose properties are initially set to unknown.

2) Let *R* be an XML root information item whose properties are set as follows:

a) The [children] property is a list with one element, *EI*.

b) The [notations] property is the empty set.

c) The [unparsed entities] property is the empty set.

d) The [version] property is “no value”.

e) The [standalone] property is “no value”.

3) Case:

a) If the <XML element> is contained within the scope of an <XML binary encoding>, then let *XBE* be the <XML binary encoding> with innermost scope that contains the <XML element>.

b) Otherwise, let *XBE* be an implementation-defined <XML binary encoding>.

4) Case:

a) If *XBE* specifies BASE64, then let *ENC* be an indication that binary strings are to be encoded in base64.

b) Otherwise, let *ENC* be an indication that binary strings are to be encoded in hex.

5) Let *i* range from 1 (one) to *n*. If the value of AV_i is not null, then:

a) Let CAV_i be the result of applying the General Rules of Subclause 9.16, "Mapping values of SQL data types to values of XML Schema data types", to AV_i, and *ENC* as the *ENCODING*, resulting in a character string CAV_i of Unicode characters.

b) Let AII_i be an XML attribute information item having the following properties:

i) The [local name] property is the XML QName local part of ANC_i.

ii) The [prefix] property is the XML QName prefix of ANC_i, if any; otherwise, "no value".

iii) The [namespace name] property is $NSURI_i$.

iv) The [normalized value] is the result of applying the rules of [XML], section 3.3.3, "Attribute Value Normalization", to CAV_i.

v) The [specified] property indicates that the attribute was specified rather than defaulted.

vi) The [attribute type] property is "CDATA".

vii) The [references] property is "no value".

viii) The [owner element] property is *EI*.

6) Let *m* be the number of occurrences of <XML element content> in <XML element> that are not the null value. Let EC_j, 1 (one) $\leq j \leq m$, be the *j*-th non-null <XML element content> contained in <XML element>.

For each *j*, 1 (one) $\leq j \leq m$:

a) Case:

i) If the declared type of EC_j is XML, then let CEC_j be a variable whose value is identical to EC_j.

ii) Otherwise, let $XMLV_j$ be the result of applying the General Rules of Subclause 9.16, "Mapping values of SQL data types to values of XML Schema data types", to EC_j, and *ENC* as the *ENCODING*. $XMLV_j$ is a character string of Unicode characters. Let CEC_j be the result of

```
XMLPARSE (CONTENT XMLVj PRESERVE WHITESPACE)
```

b) Let RII_j be the XML root information item of CEC_j. Let $LXII_j$ be the [children] property of RII_j. (Thus, $LXII_j$ is a list of individual XML information items $LXII_{j,k}$, for *k* between 1 (one) and the number of

XML information items in the list, *m*(*j*).) The [parent] property of each XML information item $LXII_{j,k}$, for *k* between 1 (one) and *m*(*j*), is set to *EI*.

i) If the [notations] property of RII_j is not empty, then it is implementation-defined whether the [notations] property of RII_j is copied to the [notations] property of *R*.

ii) If the [unparsed entities] property of RII_j is not empty, then it is implementation-defined whether the [unparsed entities] property of RII_j is copied to the [unparsed entities] property of *R*.

7) The properties of *EI* are set as follows:

   a) The [parent] property is set to *R*.

   b) The [local name] property is set to the XML QName local part of *EN*.

   c) The [prefix] property is set to “no value” if *EN* has no XML QName prefix; otherwise, it is set to the XML QName prefix of *EN*.

   d) The [namespace name] property is set to *ENSURI*.

   e) The [children] property is the list of XML information items obtained by concatenating the lists $LXII_j$, for *j* from 1 (one) through *m*, in that order.

   f) The [attributes] property is the set of XML attribute information items AII_l for all *l*, 1 (one) $\leq l \leq n$, such that AV_l is not null.

   g) The [namespace attributes] property is set to the value determined by evaluation of the General Rules of Subclause 10.9, “Determination of [namespace attributes] property”, with *EI* as the *INFOITEM*.

   h) The [base URI] property is set to an implementation-dependent value.

   i) The [in-scope namespaces] property, and of all XML element information items that are descendents of *EI*, are the values determined by evaluation of the General Rules of Subclause 10.10, “Determination of the [in-scope namespaces] property”, with *EI* as the *INFOITEM*.

8) The General Rules of Subclause 10.11, “Determination of [whitespace element content] property”, are executed, with *EI* as the *INFOITEM*.

9) The result of <XML element> is the XML value that is the collection of SQL/XML information items consisting of *R* together with every SQL/XML information item that can be reached by traversing the properties of *R*.

Conformance Rules

1) Without Feature X031, “XMLElement”, conforming SQL language shall not contain an <XML element>.

2) Without Feature X080, “Namespaces in XML publishing”, in conforming SQL language, <XML element> shall not immediately contain <XML namespace declaration>.

6.11 <XML forest>

Function

Generate an XML value with a list of XML element information items as the children of its XML root information item.

Format

```
<XML forest> ::=
    XMLFOREST <left paren> [ <XML namespace declaration> <comma> ]
    <forest element list> <right paren>

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

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

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

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

Syntax Rules

1) The scope of the <XML namespace declaration> is the <XML forest>.
2) Let *n* be the number of occurrences of <forest element> in <forest element list>. For each *i* between 1 (one) and *n*:
   a) Let F_i be the *i*-th <forest element> contained in <forest element list>.
   b) Let FV_i be the <forest element value> immediately contained in F_i. The declared type of FV_i shall be a predefined type, a distinct type, or a collection type.
   c) Case:
      i) If F_i contains a <forest element name> FEN_i, then let FNC_i be the result of the partially escaped mapping from an SQL <identifier> to an XML Name as specified in Subclause 9.1, "Mapping SQL <identifier>s to XML Names", applied to FEN_i.
      ii) Otherwise, FV_i shall be a column reference. Let CN_i be the <column name> of the column designated by FV_i. Let FNC_i be the result of the fully escaped mapping from an SQL <identifier> to an XML Name specified in Subclause 9.1, "Mapping SQL <identifier>s to XML Names", applied to CN_i.
   d) FNC_i shall be an XML QName.
   e) The Syntax Rules of Subclause 10.8, "Determination of [namespace name] property", are applied, with FNC_i as *QNAME* and with the <XML forest> as *BNFTERM*, resulting in an XML namespace URI $NSURI_i$.

Access Rules

1) If the declared type *DT* of the <value expression> *VE* immediately contained in a <forest element value> is a distinct type, then let *ST* be the source type of *DT*. The Access Rules of Subclause 6.12, "<cast specification>", in ISO/IEC 9075-2, are applied to:

```
CAST ( VE AS ST )
```

General Rules

1) Case:
   a) If the <XML forest> is contained within the scope of an <XML binary encoding>, then let *XBE* be the <XML binary encoding> with innermost scope that contains the <XML forest>.
   b) Otherwise, let *XBE* be an implementation-defined <XML binary encoding>.
2) Case:
   a) If *XBE* specifies BASE64, then let *ENC* be an indication that binary strings are to be encoded in base64.
   b) Otherwise, let *ENC* be an indication that binary strings are to be encoded in hex.
3) For each *i* between 1 (one) and *n*, let V_i be the value of the *i*-th <forest element> contained in <forest element list>.
4) Case:
   a) If every V_i, 1 (one) $\leq i \leq n$, is null, then the result of the <XML forest> is the null value.
   b) Otherwise:
      i) Let *R* be an XML root information item with the following properties:
         1) The [children] property is initially the empty list.
         2) The [notations] property is the empty set.
         3) The [unparsed entities] property is the empty set.
         4) The [standalone] property is "no value".
         5) The [version] property is "no value".
      ii) For each *i* such that V_i is not null:
         1) Let E_i be an XML element information item whose properties are initially set to unknown.
         2) Case:
            A) If the most specific type of V_i is XML, then let C_i be a variable whose value is identical to V_i.

B) Otherwise, let CS_i be result of applying the General Rules of Subclause 9.16, “Mapping values of SQL data types to values of XML Schema data types”, to V_i and *ENC* as the *ENCODING*. Let C_i be the result of

```
XMLPARSE (CONTENT CSi PRESERVE WHITESPACE)
```

3) Let RII_i be the XML root information item of C_i.

A) Let $LXII_i$ be the [children] property of RII_i. Let $m(i)$ be the number of XML information items in $LXII_i$. For k between 1 (one) and $m(i)$, let $LXII_{i,k}$ be the k-th XML information item in $LXII_i$. For all k between 1 (one) and $m(i)$, set the [parent] property of $LXII_{i,k}$ to E_i.

B) If the [notations] property of RII_i is not empty, it is implementation-defined whether the [notations] property of RII_i is copied to the [notations] property of *R*.

C) If the [unparsed entities] property of RII_i is not empty, it is implementation-defined whether the [unparsed entities] property of RII_i is copied to the [unparsed entities] property of *R*.

4) The properties of E_i are set as follows:

A) The [local name] property of E_i is set to the XML QName local part of FNC_i.

B) If FNC_i has no XML QName prefix, then the [prefix] property of E_i is set to “no value”; otherwise, it is set to the XML QName prefix of FNC_i.

C) The [children] property of E_i is set to the list of XML information items $LXII_i$.

D) The [attributes] property of E_i is set to the empty set.

E) The [namespace attributes] property of E_i is determined by applying the General Rules of Subclause 10.9, “Determination of [namespace attributes] property”.

F) The [namespace name] property of E_i is set to *NSURIi* .

G) The [in-scope namespaces] property of E_i is set by executing the General Rules of Subclause 10.10, “Determination of the [in-scope namespaces] property”, with E_i as the *INFOITEM*.

H) The [base URI] property of E_i is implementation-dependent.

I) The [parent] property of E_i is set to *R*.

5) The General Rules of Subclause 10.11, “Determination of [whitespace element content] property”, are executed, with E_i as *INFOITEM*.

iii) The [children] property of *R* is set to the list of E_i for which V_i is not null.

iv) The result of <XML forest> is the collection of SQL/XML information items consisting of *R* and every XML information item that can be reached by traversing the properties of SQL/XML information items starting from *R*.

Conformance Rules

1) Without Feature X032, “XMLForest”, conforming SQL language shall not contain an <XML forest>.

2) Without Feature X080, “Namespaces in XML publishing”, in conforming SQL language, <XML forest> shall not immediately contain <XML namespace declaration>.

6.12 <XML PI>

Function

Generate an XML processing instruction.

Format

```
<XML PI> ::=
    XMLPI <left paren> NAME <identifier>
    [ <comma> <string value expression> ] <right paren>
```

Syntax Rules

1) Let *I* be the <identifier>.

2) *I* shall not consist of three characters, the first being 'X' or 'x', the second being 'M' or 'm', and the third being 'L' or 'l'.

3) The declared type of <XML PI> is XML.

Access Rules

None.

General Rules

1) Case:

   a) If the value of the <string value expression> is the null value, then the result is the null value.

   b) Otherwise:

      i) If <string value expression> is not specified, then let *SVE* be `''` (the <character string literal> for the zero-length string); otherwise, let *SVE* be the <string value expression>.

      ii) Let *PEMI* be the result of the partially escaped mapping of an SQL <identifier> to an XML Name specified in Subclause 9.1, “Mapping SQL <identifier>s to XML Names”, applied to *I*.

      iii) If the value of

      ```
      '<?' PEMI || ' ' || SVE || '?>'
      ```

      does not conform to rule [16], “PI”, of [XML], then an exception condition is raised: *data value — invalid processing instruction*.

      iv) Let *XPIII* be an XML processing instruction information item with the following properties:

         1) The value of the [target] property is *PEMI*.

2) The value of the [content] property is the value of

```
TRIM ( LEADING FROM SVE )
```

converted to Unicode using the implementation-defined mapping from the character set of *SVE* to Unicode.

3) The value of the [base URI] property is implementation-defined.

4) The value of the [notation] property is implementation-defined.

5) The value of the [parent] information item is initially set to 'unknown'.

v) Let *XRII* be an XML root information item with the following properties:

1) The [children] property is a list with one element, *XPIII*.

2) The [notations] property is implementation-defined.

3) The [unparsed entities] property is the empty set.

4) The [version] property is 'no value'.

5) The [standalone] property is 'no value'.

vi) The [parent] property of *XPIII* is set to *XRII*.

vii) The result of the <XML PI> is the XML value consisting of *XRII* and *XPIII*.

Conformance Rules

1) Without Feature X037, “XMLPI”, conforming SQL language shall not contain an <XML PI>.

6.13 <XML parse>

Function

Perform a non-validating parse of a character string to produce an XML value.

Format

```
<XML parse> ::=
    XMLPARSE <left paren> <document or content> <string value expression>
    [ <XML whitespace option> ] <right paren>

<XML whitespace option> ::= { PRESERVE | STRIP } WHITESPACE
```

Syntax Rules

1) The declared type of <string value expression> shall be a character type.

2) The declared type of the result of <XML parse> is XML.

3) If <XML whitespace option> is not specified, then STRIP WHITESPACE is implicit.

Access Rules

None.

General Rules

1) Let *DC* be <document or content>.

2) Let *V* be the value of <string value expression>.

3) Let *WO* be the <XML whitespace option>.

4) The result of <XML parse> is determined by applying the General Rules of Subclause 10.14, “Parsing a character string as an XML value”, with *DC* as *SYNTAX*, *V* as *TEXT*, and *WO* as *OPTION*.

Conformance Rules

1) Without Feature X060, “XMLParse: CONTENT option”, in conforming SQL language, <XML parse> shall not immediately contain a <document or content> that is CONTENT.

2) Without Feature X061, “XMLParse: DOCUMENT option”, in conforming SQL language, <XML parse> shall not immediately contain a <document or content> that is DOCUMENT.

3) Without Feature X062, “XMLParse: explicit WHITESPACE option”, in conforming SQL language, <XML parse> shall not contain <XML whitespace option>.

6.14 <XML root>

Function

Create an XML value by modifying the properties of the XML root information item of another XML value.

Format

```
<XML root> ::=
    XMLROOT <left paren> <XML value expression> <comma> <XML root version>
    [ <comma> <XML root standalone> ] <right paren>

<XML root version> ::= VERSION { <string value expression> | NO VALUE }

<XML root standalone> ::= STANDALONE { YES | NO | NO VALUE }
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *V* be a variable whose value is identical to the value of the <XML value expression>.

2) The [version] property of the XML root information item of *V* is set to

   Case:

   a) If the <XML root version> contains NO VALUE, then "no value".

   b) Otherwise, the value of the <string value expression>.

3) If <XML root standalone> is specified, then the [standalone] property of the XML root information item of *V* is set to

   Case:

   a) If the <XML root standalone> contains YES, then "yes".

   b) If the <XML root standalone> contains NO, then "no".

   c) Otherwise, "no value".

4) The result of the <XML root> is *V*.

Conformance Rules

1) Without Feature X033, “XMLRoot”, conforming SQL language shall not contain <XML root>.

7 Query expressions

This Clause modifies Clause 7, “Query expressions”, in ISO/IEC 9075-2.

7.1 <query expression>

This Clause modifies Subclause 7.13, “<query expression>”, in ISO/IEC 9075-2.

Function

Specify a table.

Format

```
<with clause> ::=
    WITH [ <XML query options> ] [ <comma> ] [ [ RECURSIVE ] <with list> ]
```

Syntax Rules

1) Insert this SR A <with clause> shall specify an <XML query option> or a <with list> or both.

2) Insert this SR The scope of an <XML namespace declaration> contained in an <XML query options> immediately contained in a <with clause> is the <query expression>.

3) Insert this SR The scope of an <XML binary encoding> contained in an <XML query options> immediately contained in a <with clause> is the <query expression>.

4) Insert this SR A <with clause> shall immediately contain <comma> if and only if the <with clause> immediately contains both <XML query options> and <with list>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature X081, “Query-level XML namespace declarations”, in conforming SQL language, <with clause> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

2) Insert this CR Without Feature X131, “Query-level XMLBINARY clause”, in conforming SQL language, a <with clause> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

3) Insert this CR Without Feature X135, “XMLBINARY clause in subqueries”, in conforming SQL language, a <subquery> shall not contain an <XML query options> that contains an <XML binary encoding>.

8 Predicates

This Clause modifies Clause 8, “Predicates”, in ISO/IEC 9075-2.

8.1 <predicate>

This Clause modifies Subclause 8.1, “<predicate>”, in ISO/IEC 9075-2.

Function

Specify a condition that can be evaluated to give a boolean value.

Format

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

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 1) The result of a <predicate> is the truth value of the immediately contained <comparison predicate>, <between predicate>, <in predicate>, <like predicate>, <similar predicate>, <null predicate>, <quantified comparison predicate>, <exists predicate>, <unique predicate>, <match predicate>, <overlaps predicate>, <distinct predicate>, <member predicate>, <submultiset predicate>, <set predicate>, <type predicate>, or <XML document predicate>.

Conformance Rules

No additional Conformance Rules.

8.2 <XML document predicate>

Function

Determine whether an XML value is an XML document (that is, whether it has precisely one XML element information item in the [children] property of the XML root information). item.

Format

```
<XML document predicate> ::= <XML value expression> IS [ NOT ] DOCUMENT
```

Syntax Rules

1) Let *XVE* be the <XML value expression>.
2) The expression

   ```
   XVE IS NOT DOCUMENT
   ```

   is equivalent to

   ```
   NOT ( XVE IS DOCUMENT )
   ```

Access Rules

None.

General Rules

1) Let *V* be the value of the <XML value expression>.
2) Case:
   a) If *V* is the null value, then the result of the <XML document predicate> is *Unknown*.
   b) Otherwise, let *C* be the [children] property of the XML root information item of *V*.

      Case:

      i) If the number of XML element information items contained in *C* is one and the number of XML character information items contained in *C* is zero, then the result of the <XML document predicate> is *True*.
      ii) Otherwise, the result of the <XML document predicate> is *False*.

Conformance Rules

1) Without Feature X090, “XML document predicate”, conforming SQL language shall not contain <XML document predicate>.

This page intentionally left blank.

9 Mappings

9.1 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.

Access 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 9 — 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 $\boldsymbol{T}_i$ be the mapping of S_i to Unicode using *TM* and let $\boldsymbol{X}_i$ 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 $\boldsymbol{X_i}$ is implementation-defined.

b) If S_i is <colon>, then

Case:

i) If i = 1 (one), then let $\boldsymbol{X_i}$ be **`_x003A_`**.

ii) If *EV* is fully escaped, then let $\boldsymbol{X_i}$ be **`_x003A_`**.

iii) Otherwise, let $\boldsymbol{X_i}$ be $\boldsymbol{T_i}$.

c) If $i \leq N$–1, S_i is <underscore>, and S_{i+1} is the lowercase letter **`x`**, then let $\boldsymbol{X_i}$ be **`_x005F_`**.

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

Case:

i) If S_1 is the lowercase letter **`x`**, then let $\boldsymbol{X_1}$ be **`_x0078_`**.

ii) If S_1 is the uppercase letter **`X`**, then let $\boldsymbol{X_1}$ be **`_x0058_`**.

e) If $\boldsymbol{T_i}$ is not a valid XML NameChar, or if i = 1 (one) and $\boldsymbol{T_1}$ is not a valid first character of an XML Name, then:

i) Let $\boldsymbol{U_1}$, $\boldsymbol{U_2}$, ..., $\boldsymbol{U_8}$ be the eight <uppercase hexit>s such that $\boldsymbol{T_i}$ is U+$\boldsymbol{U_1 U_2 ... U_8}$ in the UCS-4 encoding.

ii) Case:

1) If $\boldsymbol{U_1}$ = **`0`** (zero), $\boldsymbol{U_2}$ = **`0`** (zero), $\boldsymbol{U_3}$ = **`0`** (zero), and $\boldsymbol{U_4}$ = **`0`** (zero), then let $\mathbf{X}_i$ be **`_x`**$\boldsymbol{U_5 U_6 U_7 U_8}$**`_`**.

NOTE 10 — This case implies that $\boldsymbol{T_i}$ has a UCS-2 encoding, which is U+$\boldsymbol{U_5 U_6 U_7 U_8}$.

2) Otherwise, let $\boldsymbol{X_i}$ be **`_x`**$\boldsymbol{U_3 U_4 U_5 U_6 U_7 U_8}$**`_`**.

NOTE 11 — The normative definition of valid XML Name characters is found in [XML] 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 $\boldsymbol{X_i}$ be $\boldsymbol{T_i}$.

NOTE 12 — 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.

5) Let ***`XMLN`*** be the character string concatenation of $\boldsymbol{X_1}$, $\boldsymbol{X_2}$, ..., and $\boldsymbol{X_N}$ in order from left to right.

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

Conformance Rules

None.

9.2 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) ≤ *i* ≤ *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) ≤ *j* ≤ *m*, in *S*, let NPS_j be defined as follows.

      Case:

      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) ≤ *j* ≤ *m*.

3) For each *i* between 1 (one) and *n*, let $\mathbf{XMLN}_i$ be the XML Name formed by the application of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", to $SQLI_i$ using the fully escaped variant of the mapping.

4) Let ***XMLR*** be the result of:

```
NP(XMLN1) || '.' || NP(XMLN2) || '.' || ...  || NP(XMLNn)
```

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

Conformance Rules

None.

9.3 Mapping an SQL table to XML and an XML Schema document

Function

Define the mapping of an SQL table to an XML Schema document that describes the structure of the mapped XML, and either an XML document or an XML forest of elements.

Syntax 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 to elements that are marked with **`xsi:nil="true"`** (nil). Let ***`TABLEFOREST`*** be the choice of whether to map the table to an XML forest of elements (*True*) or to an XML document with a single XML element (*False*). Let ***`TARGETNS`*** be the XML target namespace URI of the XML Schema and data to be mapped. If ***`TARGETNS`*** is the zero-length string, then no XML namespace is added. Let *DATA* be the choice of whether the mapping results in an XML representation of the data of *T* (*True*) or not (*False*). Let *METADATA* be the choice of whether the mapping results in an XML Schema document that describes the XML representation of the data of *T* (*True*) or not (*False*). Let *U* be the authorization identifier that is invoking this mapping. Let *ENCODING* be the choice of whether binary strings are to be encoded in base64 or in hex.

2) At least one of *DATA* and *METADATA* shall be *True*.

Access Rules

None.

General Rules

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

2) Let ***`XMLTN`*** be the result of applying the mapping defined in Subclause 9.1, "Mapping SQL <identifier>s to XML Names", to *TN* using the fully escaped variant of the mapping.

3) If any of the visible columns of *T* is an XML unmappable column, then a completion condition is raised: *warning — column cannot be mapped to XML*.

4) If *METADATA* is *True*, then:

   a) Let ***`XMLTYPEN`*** be the result of applying the mapping defined in Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", to "TableType", *TC*, *TS*, and *TN*.

   b) Let *CT* be the XML visible columns of *T* for *U*. Let ***`XSCT`*** be the result of applying the mapping defined in Subclause 9.10, "Mapping a collection of SQL data types to XML Schema data types", to the data types and domains of the columns of *CT* using *NULLS* as the choice of whether to map null values to absent elements or to elements that are marked with **`xsi:nil="true"`**, and *ENCODING* as the choice of whether binary strings are to be encoded using base64 or using hex.

NOTE 13 — "XML visible column" is defined in Subclause 4.8.5, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

c) Let ***XST*** be the result of applying the mapping defined in Subclause 9.6, "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"`**, ***TABLEFOREST*** as the choice of how to map *T*, and *U* as the invoker of this mapping.

d) Let ***SQLXMLNS*** be the value of the XML namespace definition provided for the XML namespace prefix **`sqlxml`** in Table 1, "XML namespace prefixes and their URIs".

e) Let ***XSDNS*** be the value of the XML namespace definition provided for the XML namespace prefix **`xsd`** in Table 1, "XML namespace prefixes and their URIs".

f) It is implementation-defined whether ***XMLDECL*** is the zero-length string or

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

or another XML declaration that conveys encoding information.

g) Let ***SLOCVAL*** be an implementation-defined URI that references the XML Schema document describing the XML namespace ***SQLXMLNS***. It is implementation-defined whether ***SLOC*** is the zero-length string or

```
schemaLocation="SLOCVAL"
```

Case:

i) If neither ***XSCT*** nor ***XST*** uses the XML namespace that corresponds to the **`sqlxml`** XML namespace prefix, then let ***XSDIMP*** be the zero-length string.

ii) If at least one of ***XSCT*** and ***XST*** uses the XML namespace that corresponds to the ***sqlxml*** XML namespace prefix, then let ***XSDIMP*** be

```
<xsd:import namespace="SQLXMLNS" SLOC />
```

h) Case:

i) If neither ***XSCT*** nor ***XST*** uses the XML namespace that corresponds to the **`sqlxml`** XML namespace prefix, then let ***XSDECL*** be the zero-length string.

ii) If at least one of ***XSCT*** and ***XST*** uses the XML namespace that corresponds to the ***sqlxml*** XML namespace prefix, then let ***XSDECL*** be

```
xmlns:sqlxml="SQLXMLNS"
```

i) Case:

i) If ***TABLEFOREST*** is <u>*False*</u> and ***TARGETNS*** is not the zero-length string, then ***XS*** has the following contents:

```
XMLDECL
<xsd:schema
      xmlns:xsd="XSDNS"
      XSDECL
      targetNamespace="TARGETNS"
      elementFormDefault="qualified">
   XSDIMP
   XSCT
   XST
   <xsd:element name="XMLTN" type="XMLTYPEN" />
</xsd:schema>
```

ii) If ***TABLEFOREST*** is <u>*False*</u> and ***TARGETNS*** is the zero-length string, then ***XS*** has the following content:

```
XMLDECL
<xsd:schema
      xmlns:xsd="XSDNS"
      XSDECL>
   XSDIMP
   XSCT
   XST
   <xsd:element name="XMLTN" type="XMLTYPEN" />
</xsd:schema>
```

iii) If ***TABLEFOREST*** is <u>*True*</u> and ***TARGETNS*** is not the zero-length string, then ***XS*** has the following contents:

```
XMLDECL
<xsd:schema
      xmlns:xsd="XSDNS"
      XSDECL
      targetNamespace="TARGETNS"
      elementFormDefault="qualified">
   XSDIMP
   XSCT
   XST
</xsd:schema>
```

iv) If ***TABLEFOREST*** is <u>*True*</u> and ***TARGETNS*** is the zero-length string, then ***XS*** has the following content:

```
XMLDECL
<xsd:schema
      xmlns:xsd="XSDNS"
      XSDECL>
   XSDIMP
   XSCT
   XST
</xsd:schema>
```

j) Let ***XSR*** be:

```
XMLSERIALIZE ( DOCUMENT
                    XMLPARSE ( DOCUMENT 'XS' PRESERVE WHITESPACE )
                    AS CLOB )
```

5) Case:

a) If *METADATA* is <u>*True*</u>, then let ***XSL*** be the URI that identifies ***XSR***.

Case:

i) If ***TARGETNS*** is the zero-length string, then let *XSLA* be

```
xsi:noNamespaceSchemaLocation="XSL"
```

ii) Otherwise, let ***XSLA*** be

```
xsi:schemaLocation="TARGETNS XSL"
```

b) Otherwise, let ***XSLA*** be the zero-length string.

6) If *DATA* is <u>*True*</u>, then:

a) Let ***XDROWS*** be the result of applying the mapping defined in Subclause 9.12, "Mapping an SQL table to an XML element or an XML forest", 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"`**, ***TABLEFOREST*** as the choice of how to map the table, ***TARGETNS*** indicating the XML target namespace, *U* as the invoker of this mapping, and *ENCODING* as the choice of whether binary strings are to be encoded using base64 or using hex.

b) Let ***XSINS*** be the value of the XML namespace definition provided for the XML namespace prefix **`xsi`** in Table 1, "XML namespace prefixes and their URIs".

c) Case:

i) If ***TABLEFOREST*** is <u>*False*</u> and ***TARGETNS*** is the zero-length string, then ***XD*** has the following contents:

```
XMLDECL
<XMLTN
   xmlns:xsi="XSINS"
   XSLA>
   XDROWS
</XMLTN>
```

ii) If ***TABLEFOREST*** is <u>*False*</u> and ***TARGETNS*** is not the zero-length string, then ***XD*** has the following contents:

```
XMLDECL
<XMLTN
   xmlns:xsi="XSINS"
   xmlns:xsd="TARGETNS"
   XSLA>
   XDROWS
</XMLTN>
```

iii) If ***TABLEFOREST*** is <u>*True*</u>, then ***XD*** is ***XDROWS***.

d) Case:

i) If ***TABLEFOREST*** is <u>*True*</u>, then let ***XDR*** be:

```
XMLSERIALIZE ( CONTENT
               XMLPARSE ( CONTENT 'XD' PRESERVE WHITESPACE )
               AS CLOB )
```

ii) If ***TABLEFOREST*** is <u>*False*</u>, then let ***XDR*** be:

```
XMLSERIALIZE ( DOCUMENT
               XMLPARSE ( DOCUMENT 'XD' PRESERVE WHITESPACE )
               AS CLOB )
```

7) If *DATA* is <u>*True*</u>, then ***XDR*** is the XML representation of the data of *T*. If *METADATA* is <u>*True*</u>, then ***XSR*** is the XML Schema document that describes the XML representation of the data of *T*.

Conformance Rules

1) Without Feature X041, “Basic table mapping: nulls absent”, a conforming application shall not invoke this Subclause of this part of this International Standard with *NULLS* set to indicate that nulls are mapped to elements that are marked to absent elements.

2) Without Feature X042, “Basic table mapping: null as nil”, a conforming application shall not invoke this Subclause of this part of this International Standard with *NULLS* set to indicate that nulls are mapped to elements that are marked with **`xsi:nil="true"`**.

3) Without Feature X043, “Basic table mapping: table as forest”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TABLEFOREST* set to <u>*True*</u>.

4) Without Feature X044, “Basic table mapping: table as element”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TABLEFOREST* set to <u>*False*</u>.

5) Without Feature X045, “Basic table mapping: with target namespace”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TARGETNS* that is not a zero-length string.

6) Without Feature X046, “Basic table mapping: data mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with with *DATA* set to <u>*True*</u>.

7) Without Feature X047, “Basic table mapping: metadata mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with *METADATA* set to <u>*True*</u>.

8) Without Feature X048, “Basic table mapping: base64 encoding of binary strings”, a conforming application shall not invoke this Subclause of this part of this International Standard with *ENCODING* set to indicate that binary strings are to be encoded using base64.

9) Without Feature X049, “Basic table mapping: hex encoding of binary strings”, a conforming application shall not invoke this Subclause of this part of this International Standard with *ENCODING* set to indicate that binary strings are to be encoded using hex.

9.4 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.

Syntax 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 to elements that are marked with **`xsi:nil="true"`** (nil). Let ***`TABLEFOREST`*** be the choice of whether to map a table to an XML forest of elements (*True*) or to a single XML element (*False*). Let ***`TARGETNS`*** be the XML target namespace URI of the XML Schema and data to be mapped. If ***`TARGETNS`*** is the zero-length string, then no XML namespace is added. Let *DATA* be the choice of whether the mapping results in an XML representation of the data of *S* (*True*) or not (*False*). Let *METADATA* be the choice of whether the mapping results in an XML Schema document that describes the XML representation of the data of *S* (*True*) or not (*False*). Let *U* be the authorization identifier that is invoking this mapping. Let *ENCODING* be the choice of whether binary strings are to be encoded using base64 or using hex.

2) At least one of *DATA* and *METADATA* shall be *True*.

Access Rules

None.

General Rules

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

2) Let ***`XMLSN`*** be the result of applying the mapping defined in Subclause 9.1, "Mapping SQL <identifier>s to XML Names", to *SN* using the fully escaped variant of the mapping.

3) If *METADATA* is *True*, then:

   a) Let *CT* be the visible columns of the viewed and base tables contained in *S* for *U*. If any of the visible columns of *CT* is an XML unmappable column, then a completion condition is raised: *warning — column cannot be mapped to XML*.

      NOTE 14 — "visible column" is defined in Subclause 4.8.5, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

   b) Let ***`XSCT`*** be the result of applying the mapping defined in Subclause 9.10, "Mapping a collection of SQL data types to XML Schema data types", to the data types and domains of the columns of *CT* using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with **`xsi:nil="true"`**, and *ENCODING* as the choice of whether binary strings are to be encoded using base64 or using hex.

c) Let ***XMLTYPEN*** be the result of applying the mapping defined in Subclause 9.2, “Mapping a multi-part SQL name to an XML Name”, to “SchemaType”, *SC*, and *SN*.

d) Let ***XST*** be the result of applying the mapping defined in Subclause 9.7, “Mapping an SQL schema to XML Schema data types”, to *S* using ***TABLEFOREST*** as the choice of how to map *T* and *U* as the invoker of this mapping.

e) Let ***SQLXMLNS*** be the value of the XML namespace definition provided for the XML namespace variable **sqlxml** in Table 1, “XML namespace prefixes and their URIs”.

f) Let ***XSDNS*** be the value of the XML namespace definition provided for the XML namespace variable **xsd** in Table 1, “XML namespace prefixes and their URIs”.

g) It is implementation-defined whether ***XMLDECL*** is the zero-length string or

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

or another XML declaration that conveys encoding information.

h) Let ***SLOCVAL*** be an implementation-defined URI that references the XML Schema document describing the XML namespace ***SQLXMLNS***. It is implementation-defined whether ***SLOC*** is the zero-length string or

```
schemaLocation="SLOCVAL"
```

Case:

i) If neither ***XSCT*** nor ***XST*** uses the XML namespace that corresponds to the **sqlxml** XML namespace prefix, then let ***XSDIMP*** be the zero-length string.

ii) If at least one of ***XSCT*** and ***XST*** uses the XML namespace that corresponds to the ***sqlxml*** XML namespace prefix, then let ***XSDIMP*** be

```
<xsd:import
    namespace="SQLXMLNS" SLOC />
```

i) Case:

i) If neither ***XSCT*** nor ***XST*** uses the XML namespace that corresponds to the **sqlxml** XML namespace prefix, then let ***XSDECL*** be the zero-length string.

ii) If at least one of ***XSCT*** and ***XST*** uses the XML namespace that corresponds to the ***sqlxml*** XML namespace prefix, then let ***XSDECL*** be

```
xmlns:sqlxml="SQLXMLNS"
```

j) Case:

i) If ***TARGETNS*** is the zero-length string, then ***XS*** has the following contents:

```
XMLDECL
<xsd:schema
      xmlns:xsd="XSDNS"
      XSDECL>
   XSDIMP
   XSCT
   XST
   <xsd:element name="XMLSN" type="XMLTYPEN" />
</xsd:schema>
```

ii) If ***TARGETNS*** is not the zero-length string, then ***XS*** has the following contents:

```
XMLDECL
<xsd:schema
      xmlns:xsd="XSDNS"
      XSDECL
      targetNamespace="TARGETNS"
      elementFormDefault="qualified">
   XSDIMP
   XSCT
   XST
   <xsd:element name="XMLSN" type="XMLTYPEN" />
</xsd:schema>
```

k) Let ***XSR*** be:

```
XMLSERIALIZE ( DOCUMENT
               XMLPARSE ( DOCUMENT 'XS' PRESERVE WHITESPACE )
               AS CLOB )
```

4) Case:

a) If *METADATA* is *True*, then let ***XSL*** be the URI that identifies ***XSR***.

Case:

i) If ***TARGETNS*** is the zero-length string, then let *XSLA* be

```
xsi:noNamespaceSchemaLocation="XSL"
```

ii) Otherwise, let *XSLA* be

```
xsi:schemaLocation="TARGETNS XSL"
```

b) Otherwise, let *XSLA* be the zero-length string.

5) If *DATA* is *True*, then:

a) Let ***XDSCHEMA*** be the result of applying the mapping defined in Subclause 9.13, "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"`**, ***TABLEFOREST*** as the choice of how to map the tables, ***TARGETNS*** indicating the XML target namespace, *U* as the invoker of this mapping, and *ENCODING* as the choice of whether binary strings are to be encoded using base64 or using hex.

b) Let ***XSINS*** be the value of the XML namespace definition provided for the XML namespace variable **`xsi`** in Table 1, "XML namespace prefixes and their URIs".

c) Case:

i) If ***TARGETNS*** is the zero-length string, then ***XD*** has the following contents:

```
XMLDECL
<XMLSN
   xmlns:xsi="XSINS"
   XSLA>
   XDSCHEMA
</XMLSN>
```

ii) If ***TARGETNS*** is not the zero-length string, then ***XD*** has the following contents:

```
XMLDECL
<XMLSN
   xmlns:xsi="XSINS"
   xmlns="TARGETNS"
   XSLA>
   XDSCHEMA
</XMLSN>
```

d) Let ***XDR*** be:

```
XMLSERIALIZE ( DOCUMENT
               XMLPARSE ( DOCUMENT 'XD' PRESERVE WHITESPACE )
               AS CLOB )
```

6) If *DATA* is True, then ***XDR*** is the XML representation of the data of *S*. If *METADATA* is True, then ***XSR*** is the XML Schema document that describes the XML representation of the data of *S*.

Conformance Rules

1) Without Feature X051, “Advanced table mapping: nulls absent”, a conforming application shall not invoke this Subclause of this part of this International Standard with *NULLS* set to indicate that nulls are mapped to elements that are marked to absent elements.

2) Without Feature X052, “Advanced table mapping: null as nil”, a conforming application shall not invoke this Subclause of this part of this International Standard with *NULLS* set to indicate that nulls are mapped to elements that are marked with **`xsi:nil="true"`**.

3) Without Feature X053, “Advanced table mapping: table as forest”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TABLEFOREST* set to True.

4) Without Feature X054, “Advanced table mapping: table as element”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TABLEFOREST* set to False.

5) Without Feature X055, “Advanced table mapping: with target namespace”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TARGETNS* that is not a zero-length string.

6) Without Feature X056, “Advanced table mapping: data mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with with *DATA* set to *<u>True</u>*.

7) Without Feature X057, “Advanced table mapping: metadata mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with *METADATA* set to *<u>True</u>*.

8) Without Feature X058, “Advanced table mapping: base64 encoding of binary strings”, a conforming application shall not invoke this Subclause of this part of this International Standard with *ENCODING* set to indicate that binary strings are to be encoded using base64.

9) Without Feature X059, “Advanced table mapping: hex encoding of binary strings”, a conforming application shall not invoke this Subclause of this part of this International Standard with *ENCODING* set to indicate that binary strings are to be encoded using hex.

9.5 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.

Syntax 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 to elements that are marked with `xsi:nil="true"` (nil). Let ***`TABLEFOREST`*** be the choice of whether to map a table to an XML forest of elements (*True*) to a single XML element (*False*). Let ***`TARGETNS`*** be the XML target namespace URI of the XML Schema and data to be mapped. If ***`TARGETNS`*** is the zero-length string, then no XML namespace is added. Let *DATA* be the choice of whether the mapping results in an XML representation of the data of *C* (*True*) or not (*False*). Let *METADATA* be the choice of whether the mapping results in an XML Schema document that describes the XML representation of the data of *C* (*True*) or not (*False*). Let *U* be the authorization identifier that is invoking this mapping. Let *ENCODING* be the choice of whether binary strings are to be encoded using base64 or using hex.
2) At least one of *DATA* and *METADATA* shall be *True*.

Access Rules

None.

General Rules

1) Let *CN* be the <catalog name> of *C*.
2) Let ***`XMLCN`*** be the result of applying the mapping defined in Subclause 9.1, "Mapping SQL <identifier>s to XML Names", to *CN* using the fully escaped variant of the mapping.
3) If *METADATA* is *True*, then:
   a) Let *CT* be the visible columns of the viewed and base tables contained in *C* for *U*. If any of the visible columns of *CT* is an XML unmappable column, then a completion condition is raised: *warning — column cannot be mapped to XML*.

      NOTE 15 — "visible column" is defined in Subclause 4.8.5, "Visibility of columns, tables, and schemas in mappings from SQL to XML".
   b) Let ***`XSCT`*** be the result of applying the mapping defined in Subclause 9.10, "Mapping a collection of SQL data types to XML Schema data types", to the data types and domains of the columns of *CT* using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with `xsi:nil="true"`, and *ENCODING* as the choice of whether binary strings are to be encoded using base64 or using hex.

c) Let ***XMLTYPEN*** be the result of applying the mapping defined in Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", to "CatalogType" and *CN*.

d) Let ***XST*** be the result of applying the mapping defined in Subclause 9.8, "Mapping an SQL catalog to XML Schema data types", to *C* using *U* as the invoker of this mapping.

e) Let ***SQLXMLNS*** be the value of the XML namespace definition provided for the XML namespace prefix **sqlxml** in Table 1, "XML namespace prefixes and their URIs".

f) Let ***XSDNS*** be the value of the XML namespace definition provided for the XML namespace prefix **xsd** in Table 1, "XML namespace prefixes and their URIs".

g) It is implementation-defined whether ***XMLDECL*** is the zero-length string or

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

or another XML declaration that conveys encoding information.

h) Let ***SLOCVAL*** be an implementation-defined URI that references the XML Schema document describing the XML namespace ***SQLXMLNS***. It is implementation-defined whether ***SLOC*** is the zero-length string or

```
schemaLocation="SLOCVAL"
```

Case:

i) If neither ***XSCT*** nor ***XST*** uses the XML namespace that corresponds to the **sqlxml** XML namespace prefix, then let ***XSDIMP*** be the zero-length string.

ii) If at least one of ***XSCT*** and ***XST*** uses the XML namespace that corresponds to the ***sqlxml*** XML namespace prefix, then let ***XSDIMP*** be

```
<xsd:import
    namespace="SQLXMLNS" SLOC />
```

i) Case:

i) If neither ***XSCT*** nor ***XST*** uses the XML namespace that corresponds to the **sqlxml** XML namespace prefix, then let ***XSDECL*** be the zero-length string.

ii) If at least one of ***XSCT*** and ***XST*** uses the XML namespace that corresponds to the ***sqlxml*** XML namespace prefix, then let ***XSDECL*** be

```
xmlns:sqlxml="SQLXMLNS"
```

j) Case:

i) If ***TARGETNS*** is the zero-length string, then ***XS*** has the following contents:

```
XMLDECL
<xsd:schema
      xmlns:xsd="XSDNS"
      XSDECL>
   XSDIMP
   XSCT
   XST
   <xsd:element name="XMLCN" type="XMLTYPEN" />
</xsd:schema>
```

ii) If ***TARGETNS*** is not the zero-length string, then ***XS*** has the following contents:

```
XMLDECL
<xsd:schema
      xmlns:xsd="XSDNS"
      XSDECL
      targetNamespace="TARGETNS"
      elementFormDefault="qualified">
   XSDIMP
   XSCT
   XST
   <xsd:element name="XMLCN" type="XMLTYPEN" />
</xsd:schema>
```

k) Let ***XSR*** be:

```
XMLSERIALIZE ( DOCUMENT
               XMLPARSE ( DOCUMENT 'XS' PRESERVE WHITESPACE )
               AS CLOB )
```

4) Case:

a) If *METADATA* is <u>*True*</u>, then let ***XSL*** be the URI that identifies ***XSR***.

Case:

i) If ***TARGETNS*** is the zero-length string, then let *XSLA* be

```
xsi:noNamespaceSchemaLocation="XSL"
```

ii) Otherwise, let *XSLA* be

```
xsi:schemaLocation="TARGETNS XSL"
```

b) Otherwise, let *XSLA* be the zero-length string.

5) If *DATA* is <u>*True*</u>, then:

a) Let ***XDCATALOG*** be the result of applying the mapping defined in Subclause 9.14, "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 to elements that are marked with **`xsi:nil="true"`**, ***TABLEFOREST*** as the choice of how to map tables, ***TARGETNS*** indicating the XML target namespace, *U* as the invoker of this mapping and *ENCODING* as the choice of whether binary strings are to be encoded using base64 or using hex.

b) Let ***XSINS*** be the value of the XML namespace definition provided for the XML namespace prefix **`xsi`** in Table 1, "XML namespace prefixes and their URIs".

c) Case:

i) If ***TARGETNS*** is the zero-length string, then ***XD*** has the following contents:

```
XMLDECL
<XMLCN
   xmlns:xsi="XSINS"
   XSLA>
   XDCATALOG
</XMLCN>
```

ii) If ***TARGETNS*** is not the zero-length string, then ***XD*** has the following contents:

```
XMLDECL
<XMLCN
   xmlns:xsi="XSINS"
   xmlns="TARGETNS"
   XSLA>
   XDCATALOG
</XMLCN>
```

d) Let ***XDR*** be:

```
XMLSERIALIZE ( DOCUMENT
               XMLPARSE ( DOCUMENT 'XD' PRESERVE WHITESPACE )
               AS CLOB )
```

6) If *DATA* is *True*, then ***XDR*** is the XML representation of the data of *C*. If *METADATA* is *True*, then ***XSR*** is the XML Schema document that describes the XML representation of the data of *C*.

Conformance Rules

1) Without Feature X051, “Advanced table mapping: nulls absent”, a conforming application shall not invoke this Subclause of this part of this International Standard with *NULLS* set to indicate that nulls are mapped to elements that are marked to absent elements.

2) Without Feature X052, “Advanced table mapping: null as nil”, a conforming application shall not invoke this Subclause of this part of this International Standard with *NULLS* set to indicate that nulls are mapped to elements that are marked with **`xsi:nil="true"`**.

3) Without Feature X053, “Advanced table mapping: table as forest”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TABLEFOREST* set to *True*.

4) Without Feature X054, “Advanced table mapping: table as element”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TABLEFOREST* set to *False*.

5) Without Feature X055, “Advanced table mapping: with target namespace”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TARGETNS* that is not a zero-length string.

6) Without Feature X056, “Advanced table mapping: data mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with with *DATA* set to *True*.

7) Without Feature X057, "Advanced table mapping: metadata mapping", a conforming application shall not invoke this Subclause of this part of this International Standard with *METADATA* set to *True*.

8) Without Feature X058, "Advanced table mapping: base64 encoding of binary strings", a conforming application shall not invoke this Subclause of this part of this International Standard with *ENCODING* set to indicate that binary strings are to be encoded using base64.

9) Without Feature X059, "Advanced table mapping: hex encoding of binary strings", a conforming application shall not invoke this Subclause of this part of this International Standard with *ENCODING* set to indicate that binary strings are to be encoded using hex.

9.6 Mapping an SQL table to XML Schema data types

Function

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

Syntax Rules

None.

Access Rules

None.

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 ***`TABLEFOREST`*** be the choice of whether to map the table to an XML forest of elements (*True*) or to map the table to an XML documents with a single root element (*False*). 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 9.1, "Mapping SQL <identifier>s to XML Names", to *TN* using the fully escaped variant of the mapping.

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

   NOTE 16 — "XML visible column" is defined in Subclause 4.8.5, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

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

   a) Let C_i be the *i*-th XML 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 9.1, "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 9.9, "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 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 9.1, "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 9.1, "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-defined 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 9.2, “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 9.2, “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 9.1, “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) Case:

a) If ***TABLEFOREST*** is *False*, then let ***XMLTYPMAP*** 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>
```

b) If ***TABLEFOREST*** is *True*, then let ***XMLTYPMAP*** be:

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

```
         XMLCE_n
      </xsd:sequence>
   </xsd:complexType>
   <xsd:element name="XMLTN" type="XMLROWN" />
```

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

Conformance Rules

None.

9.7 Mapping an SQL schema to XML Schema data types

Function

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

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *S* be the schema provided for an application of this mapping. Let ***TABLEFOREST*** be the choice of whether to map the table to an XML forest of elements (*True*) or to map the table to an XML documents with a single root element (*False*). 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 9.1, "Mapping SQL <identifier>s to XML Names", to *SN* using the fully escaped variant of the mapping.

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

   NOTE 17 — "XML visible table" is defined in Subclause 4.8.5, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

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

   a) Let T_i be the *i*-th XML visible table of *S* for *U* in the implementation-dependent repeatable ordering 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 9.1, "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 9.2, "Mapping a multi-part SQL name to an XML Name", to "TableType", *SC*, *SN*, and *TN*.

   e) Let ***XMLROWN*** be the result of applying the mapping defined in Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", to "RowType", *SC*, *SN*, and *TN*.

   f) Case:

      i) If ***TABLEFOREST*** is *False*, then let $\boldsymbol{XMLTE_i}$ be

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

ii) If ***TABLEFOREST*** is <u>*True*</u>, then let ***XMLTE$_i$*** be

```
<xsd:element name="XMLTN" type="XMLROWN"
    minOccurs="0" maxOccurs="unbounded" />
```

6) Let ***XMLTYPEN*** be the result of applying the mapping defined in Subclause 9.2, “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 9.1, “Mapping SQL <identifier>s to XML Names”, to *SC* using the fully escaped variant of the mapping.

8) It is implementation-defined 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) Case:

a) If ***TABLEFOREST*** is <u>*False*</u>, then let ***XMLSCHEMAT*** be:

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

b) If ***TABLEFOREST*** is <u>*True*</u>, then let ***XMLSCHEMAT*** be:

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

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

Conformance Rules

None.

9.8 Mapping an SQL catalog to XML Schema data types

Function

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

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *C* be the catalog provided for an application of this mapping. 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 9.1, "Mapping SQL <identifier>s to XML Names", to *CN* using the fully escaped variant of the mapping.

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

   NOTE 18 — "XML visible schema" is defined in Subclause 4.8.5, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

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

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

   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 9.1, "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 9.2, "Mapping a multi-part SQL name to an XML Name", to "SchemaType", *CN*, and *SN*.

   e) Let $\boldsymbol{XMLSE_i}$ be

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

6) Let ***XMLTYPEN*** be the result of applying the mapping defined in Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", to "CatalogType" and *CN*.

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

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

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

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

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

Conformance Rules

None.

9.9 Mapping an SQL data type to an XML Name

Function

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

Syntax Rules

None.

Access Rules

None.

General Rules

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

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

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

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

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

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

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

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

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

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

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

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

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

Conformance Rules

None.

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

Function

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

Syntax Rules

None.

Access Rules

None.

General Rules

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

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

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

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

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

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

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

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

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

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

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

   b) Let ***`XMLN`***$_i$ be the result of applying the mapping defined in Subclause 9.9, "Mapping an SQL data type to an XML Name", to D_i.

   c) Let ***`XMLT`***$_i$ be the XML Schema data type that is the result of applying the mapping defined in Subclause 9.11, "Mapping an SQL data type to a named XML Schema data type", to D_i using *NULLS* as the choice of whether to map null values to absent elements or elements that are marked with

xsi:nil="true" and *ENCODING* as the choice of whether to encode binary strings in base64 or in hex.

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

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

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

      ```
      XMLD || XMLTi
      ```

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

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

Conformance Rules

None.

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

Function

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

Syntax Rules

None.

Access Rules

None.

General Rules

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

2) Let ***`XMLN`*** be the result of applying the mapping defined in Subclause 9.9, "Mapping an SQL data type to an XML Name", to *D*.

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

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

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

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

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

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

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

   e) Case:

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

```
<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>
```

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

   a) Let ***XMLT*** be the XML Schema data type that is the result of applying the mapping defined in Subclause 9.15, “Mapping SQL data types to XML Schema data types”, to *D* using *NULLS* as the choice of whether to to map null values to absent elements or elements that are marked with **`xsi:nil="true"`** and *ENCODING* as the choice of whether to encode binary strings in base64 or in hex.

   b) Case:

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

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

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

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

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

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

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

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

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

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

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

Conformance Rules

None.

9.12 Mapping an SQL table to an XML element or an XML forest

Function

Define the mapping of an SQL table to XML.

Syntax Rules

None.

Access Rules

None.

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 to elements that are marked with **`xsi:nil="true"`** (nil). Let ***`TABLEFOREST`*** be the choice of whether to map the table to an XML forest of elements (*True*) or to an XML document with a single root element (*False*). Let ***`TARGETNS`*** be the XML target namespace URI of the XML Schema and data to be mapped. If ***`TARGETNS`*** is the zero-length string, then no XML namespace is added. Let *U* be the authorization identifier that is invoking this mapping. Let *ENCODING* be the choice of whether to encode binary strings in base64 or in hex.

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 9.1, "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 XML visible columns of *T* for *U*.

   NOTE 19 — "XML visible column" is defined in Subclause 4.8.5, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

5) Let ***`XSINS`*** be the value of the XML namespace definition provided for the XML namespace prefix **`xsi`** in Table 1, "XML namespace prefixes and their URIs".

   Case:

   a) If *NULLS* is absent, then let ***`XSI`*** be the zero-length string.

   b) If *NULLS* is nil, let ***`XSI`*** be

   ```
   xmlns:xsi="XSINS"
   ```

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

   a) Let R_i be the *i*-th row of *T*, in the implementation-dependent repeatable ordering of *T*.

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

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

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

iii) Let $\mathbf{XMLCN}_j$ be the result of applying the mapping defined in Subclause 9.1, “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 for R_i.

v) Case:

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

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

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

3) Otherwise:

A) Let $\mathbf{XMLV}_j$ be the result of applying the mapping defined in Subclause 9.16, “Mapping values of SQL data types to values of XML Schema data types”, using V_j as the SQL data value *DV*, *NULLS* as the choice of whether to map null values to absent elements (absent) or to elements that are marked with **xsi:nil="true"** (nil), and *ENCODING* as the choice of whether to encode binary strings in base64 or in hex.

B) $\mathbf{XMLC}_j$ is

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

c) Case:

i) If ***TABLEFOREST*** is <u>*False*</u>, then let $\mathbf{XMLR}_i$ be

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

ii) If ***TABLEFOREST*** is <u>*True*</u> and ***TARGETNS*** is the zero-length string, then let $\mathbf{XMLR}_i$ be

```
<XMLTN XSI">
   XMLC1
   ...
   XMLCm
</XMLTN>
```

iii) If ***TABLEFOREST*** is <u>*True*</u> and ***TARGETNS*** is not the zero-length string, then let $\mathbf{XMLR}_i$ be

```
<XMLTN XSI xmlns="TARGETNS">
   XMLC1
```

```
      ...
      XMLCm
   </XMLTN>
```

7) Case:

   a) If ***TABLEFOREST*** is <u>*False*</u> and ***TARGETNS*** is the zero-length string, then let ***XMLTE*** be:

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

   b) If ***TABLEFOREST*** is <u>*False*</u> and ***TARGETNS*** is not the zero-length string, then let ***XMLTE*** be:

```
<XMLTN xmlns="TARGETNS">
   XMLR1
   ...
   XMLRn
</XMLTN>
```

   c) If ***TABLEFOREST*** is <u>*True*</u>, then let ***XMLTE*** be:

```
   XMLR1
   ...
   XMLRn
```

8) ***XMLTE*** is the XML that is the result of the application of this Subclause.

Conformance Rules

None.

9.13 Mapping an SQL schema to an XML element

Function

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

Syntax Rules

None.

Access Rules

None.

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 to elements that are marked with **`xsi:nil="true"`** (nil). Let ***`TABLEFOREST`*** be the choice of whether to map a table to an XML forest of elements (*True*) or to a single XML element (*False*). Let ***`TARGETNS`*** be the XML target namespace URI of the XML Schema and data to be mapped. If ***`TARGETNS`*** is the zero-length string, then no XML namespace is added. Let *U* be the authorization identifier that is invoking this mapping. Let *ENCODING* be the choice of whether to encode binary strings in base64 or in hex.

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

3) Let ***`XMLSN`*** be the result of applying the mapping defined in Subclause 9.1, "Mapping SQL <identifier>s to XML Names", to *SN* using the fully escaped variant of the mapping.

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

   NOTE 20 — "XML visible table " is defined in Subclause 4.8.5, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

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

   a) Let T_i be the *i*-th XML visible table of *S* for *U* in the implementation-dependent repeatable ordering of *S*.

   b) Let ***`XMLT`***$_i$ be the result of applying the mapping defined in Subclause 9.12, "Mapping an SQL table to an XML element or an XML forest", to T_i using *NULLS* as the choice of whether to map null values to absent elements or to elements that are marked with **`xsi:nil="true"`**, ***`TABLEFOREST`*** as the choice of how to map the table, ***`TARGETNS`*** set to the zero-length string for the application of Subclause 9.12, "Mapping an SQL table to an XML element or an XML forest", *U* as the invoker of this mapping, and *ENCODING* as the choice of whether to encode binary strings in base64 or in hex.

6) Case:

   a) If ***`TARGETNS`*** is the zero-length string, then let ***`XMLSE`*** be:

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

b) If ***TARGETNS*** is not the zero-length string, then let ***XMLSE*** be:

```
<XMLSN xmlns="TARGETNS">
   XMLT1
   ...
   XMLTn
</XMLSN>
```

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

Conformance Rules

None.

9.14 Mapping an SQL catalog to an XML element

Function

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

Syntax Rules

None.

Access Rules

None.

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 to elements that are marked with **`xsi:nil="true"`** (nil). Let ***`TABLEFOREST`*** be the choice of whether to map a table to an XML forest of elements (*True*) or to a single XML element (*False*). Let ***`TARGETNS`*** be the XML target namespace URI of the XML Schema and data to be mapped. If ***`TARGETNS`*** is the zero-length string, then no XML namespace is added. Let *U* be the authorization identifier that is invoking this mapping. Let *ENCODING* be the choice of whether to encode binary strings in base64 or in hex.

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

3) Let ***`XMLCN`*** be the result of applying the mapping defined in Subclause 9.1, "Mapping SQL <identifier>s to XML Names", to *CN* using the fully escaped variant of the mapping.

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

   NOTE 21 — "XML visible schema" is defined in Subclause 4.8.5, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

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

   a) Let S_i be the *i*-th XML visible schema of *C* for *U* in the implementation-dependent repeatable ordering of *C*.

   b) Let ***`XMLS`***$_i$ be the result of applying the mapping defined in Subclause 9.13, "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 to elements that are marked with **`xsi:nil="true"`**, ***`TABLEFOREST`*** as the choice of how to map a table, ***`TARGETNS`*** set to the zero-length string for the application of Subclause 9.13, "Mapping an SQL schema to an XML element", *U* as the invoker of this mapping, and *ENCODING* as the choice of whether to encode binary strings in base64 or in hex.

6) Case:

   a) If ***`TARGETNS`*** is the zero-length string, then let ***`XMLCE`*** be:

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

b) If ***TARGETNS*** is not the zero-length string, then let ***XMLCE*** be:

```
<XMLCN xmlns="TARGETNS">
   XMLS1
   ...
   XMLSn
</XMLCN>
```

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

Conformance Rules

None.

9.15 Mapping SQL data types to XML Schema data types

Function

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

Syntax Rules

None.

Access Rules

None.

General Rules

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

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

   Case:

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

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

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

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

5) Let **`xsd`** be the XML namespace prefix to be used to identify the XML Schema namespace as shown in Table 1, "XML namespace prefixes and their URIs".

6) Let **`sqlxml`** be the XML namespace prefix to be used to identify the XML namespace as shown in Table 1, "XML namespace prefixes and their URIs".

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

8) Case:

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

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

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

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

iv) Case:

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

A) Case:

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

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

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

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

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

```
name="CHAR"
```

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

```
length="NLIT"
```

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

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

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

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

```
name="VARCHAR"
```

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

```
maxLength="NLIT"
```

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

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

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

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

```
name="CLOB"
```

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

```
maxLength="NLIT"
```

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

```
characterSetName="SQLCSNLIT"
```

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

```
collation="SQLCONLIT"
```

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

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

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

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

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

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

ii) Case:

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

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

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

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

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

```
name="BLOB"
```

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

```
maxLength="NLIT"
```

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

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

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

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

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

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

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

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

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

iii) Case:

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

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

```
name="NUMERIC"
```

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

```
precision="PLIT"
```

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

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

```
name="DECIMAL"
```

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

```
userPrecision="UPLIT"
```

NOTE 22 — *UP* may be less than *P*, as specified in SR 22) of Subclause 6.1, “<data type>”, in ISO/IEC 9075-2.

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

```
scale="SLIT"
```

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

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

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

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

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

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

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

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

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

iii) Case:

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

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

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

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

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

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

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

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

or determined by

Case:

1) If *MAX* is equal to 2^{63}-1 (9,223,372,036,854,775,807) and *MIN* is equal to -2^{63} (-9,223,372,036,854,775,808), then *REST* is:

```
<xsd:restriction base="xsd:long"/>
```

2) If *MAX* is less than 2^{63}-1 (9,223,372,036,854,775,807) and *MIN* is equal to -2^{63} (-9,223,372,036,854,775,808), then *REST* is:

```
<xsd:restriction base="xsd:long">
  FACETMAX
</xsd:restriction>
```

3) If *MAX* is equal to 2^{63}-1 (9,223,372,036,854,775,807) and *MIN* is greater than -2^{63} (-9,223,372,036,854,775,808) but not equal to 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:long">
  FACETMIN
</xsd:restriction>
```

4) If *MAX* is less than 2^{63}-1 (9,223,372,036,854,775,807) but greater than 2^{31}-1 (2,147,483,647) and *MIN* is less than 0 (zero) or *MIN* is greater than -2^{63} (-9,223,372,036,854,775,808) but less than -2^{31} (-2,147,483,648), then *REST* is:

```
<xsd:restriction base="xsd:long">
  FACETMAX
  FACETMIN
</xsd:restriction>
```

5) If *MAX* is equal to 2^{31}-1 (2,147,483,647) and *MIN* is equal to -2^{31} (-2,147,483,648), then *REST* is:

```
<xsd:restriction base="xsd:int"/>
```

6) If *MAX* is less than 2^{31}-1 (2,147,483,647) and *MIN* is equal to -2^{31} (-2,147,483,648), then *REST* is:

```
<xsd:restriction base="xsd:int">
  FACETMAX
</xsd:restriction>
```

7) If *MAX* is equal to 2^{31}-1 (2,147,483,647) and *MIN* is greater than -2^{31} (-2,147,483,648), then *REST* is:

```
<xsd:restriction base="xsd:int">
  FACETMIN
</xsd:restriction>
```

8) If *MAX* is less than 2^{31}-1 (2,147,483,647) but greater than 2^{15}-1 (32,767) and *MIN* is less than 0 (zero) or *MIN* is greater than -2^{31} (-2,147,483,648) but less than -2^{15} (-32,786), then *REST* is:

```
<xsd:restriction base="xsd:int">
  FACETMAX
  FACETMIN
</xsd:restriction>
```

9) If *MAX* is equal to 2^{15}-1 (32,767) and *MIN* is equal to -2^{15} (-32,786), then *REST* is:

```
<xsd:restriction base="xsd:short"/>
```

10) If *MAX* is less than 2^{15}-1 (32,767) and *MIN* is equal to -2^{15} (-32,786), then *REST* is:

```
<xsd:restriction base="xsd:short">
  FACETMAX
</xsd:restriction>
```

11) If *MAX* is equal to 2^{15}-1 (32,767) and *MIN* is greater than -2^{15} (-32,786) but not equal to 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:short">
  FACETMIN
</xsd:restriction>
```

12) If *MAX* is less than 2^{15}-1 (32,767) but greater than 255 and *MIN* is less than 0 (zero) or *MIN* is greater than -2^{15} (-32,786) but less than 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:short">
  FACETMAX
  FACETMIN
</xsd:restriction>
```

13) If *MAX* is equal to 2^7 (127) and *MIN* is equal to -2^7 (-128), then *REST* is:

```
<xsd:restriction base="xsd:byte"/>
```

14) If *MAX* is less than 2^7-1 (127) and *MIN* is equal to -2^7 (-128), then *REST* is:

```
<xsd:restriction base="xsd:byte">
  FACETMAX
</xsd:restriction>
```

15) If *MAX* is equal to 2^7-1 (127) and *MIN* is greater than -2^7 (-128) but not equal to 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:byte">
  FACETMIN
</xsd:restriction>
```

16) If *MAX* is less than 2^7-1 (127) but greater than 0 (zero) and *MIN* is less than 0 (zero) or *MIN* is greater than -2^7 (-128) but less than 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:byte">
  FACETMAX
  FACETMIN
</xsd:restriction>
```

17) If *MAX* is equal to 2^{64}-1 (18,446,744,073,709,551,615) and *MIN* is equal to 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:unsignedLong"/>
```

18) If *MAX* is equal to 2^{64}-1 (18,446,744,073,709,551,615) and *MIN* is greater than 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:unsignedLong">
```

19) If *MAX* is less than 2^{31}-1 (2,147,483,647) but greater than 2^{15}-1 (32,767) and *MIN* is less than 0 (zero) or *MIN* is greater than -2^{31} (-2,147,483,648) but less than -2^{15} (-32,786), then *REST* is:

```
<xsd:restriction base="xsd:int">
  FACETMAX
  FACETMIN
</xsd:restriction>
```

20) If *MAX* is equal to 2^{15}-1 (32,767) and *MIN* is equal to -2^{15} (-32,786), then *REST* is:

```
<xsd:restriction base="xsd:short"/>
```

21) If *MAX* is less than 2^{15}-1 (32,767) and *MIN* is equal to -2^{15} (-32,786), then *REST* is:

```
<xsd:restriction base="xsd:short">
  FACETMAX
</xsd:restriction>
```

22) If *MAX* is equal to 2^{15}-1 (32,767) and *MIN* is greater than -2^{15} (-32,786) but not equal to 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:short">
  FACETMIN
</xsd:restriction>
```

23) If *MAX* is less than 2^{15}-1 (32,767) but greater than 255 and *MIN* is less than 0 (zero) or *MIN* is greater than -2^{15} (-32,786) but less than 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:short">
  FACETMAX
  FACETMIN
</xsd:restriction>
```

24) If *MAX* is equal to 2^7-1 (127) and *MIN* is equal to -2^7 (-128), then *REST* is:

```
<xsd:restriction base="xsd:byte"/>
```

25) If *MAX* is less than 2^7-1 (127) and *MIN* is equal to -2^7 (-128), then *REST* is:

```
<xsd:restriction base="xsd:byte">
  FACETMAX
</xsd:restriction>
```

26) If *MAX* is equal to 2^7-1 (127) and *MIN* is greater than -2^7 (-128) but not equal to 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:byte">
  FACETMIN
</xsd:restriction>
```

27) If *MAX* is less than 2^7-1 (127) but greater than 0 (zero) and *MIN* is less than 0 (zero) or *MIN* is greater than -2^7 (-128) but less than 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:byte">
  FACETMAX
  FACETMIN
</xsd:restriction>
```

28) If *MAX* is equal to 2^{64}-1 (18,446,744,073,709,551,615) and *MIN* is equal to 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:unsignedLong"/>
```

29) If *MAX* is equal to 2^{64}-1 (18,446,744,073,709,551,615) and *MIN* is greater than 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:unsignedLong"/>
```

30) If *MAX* is equal to 255 and *MIN* is equal to 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:unsignedByte"/>
```

31) If *MAX* is equal to 255 and *MIN* is greater than 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:unsignedByte">
  FACETMIN
</xsd:restriction>
```

32) If *MAX* is less than 255 but greater than 0 (zero) and *MIN* is equal to 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:unsignedByte">
  FACETMAX
</xsd:restriction>
```

33) If *MAX* is less than 255 but greater than 0 (zero) and *MIN* is greater than 0 (zero), then *REST* is:

```
<xsd:restriction base="xsd:unsignedByte">
  FACETMAX
  FACETMIN
</xsd:restriction>
```

34) Otherwise, *REST* is:

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

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

```
<xsd:simpleType>
  ANN
  REST
  </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-defined whether the XML text ***ANNT*** is the zero-length string or:

```
name="REAL"
```

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

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

3) Otherwise:

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

```
name="FLOAT"
```

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

```
userPrecision="UPLIT"
```

NOTE 23 — *UP* may be less than *P*, as specified in SR 24) 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-defined whether the XML text ***ANNP*** is the zero-length string or:

```
precision="PLIT"
```

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

```
minExponent="MINLIT"
```

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

```
maxExponent="MAXLIT"
```

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

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

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

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

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

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

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

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

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

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

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

ii) Case:

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

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

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

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

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

```
name="TIME"
```

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

```
scale="SLIT"
```

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

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

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

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

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

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

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

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

iii) Case:

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

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

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

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

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

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

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

```
scale="SLIT"
```

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

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

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

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

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

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

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

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

iii) Case:

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

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

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

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

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

```
name="TIMESTAMP"
```

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

```
scale="SLIT"
```

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

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

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

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

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

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

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

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

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

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

iv) Case:

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

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

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

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

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

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

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

```
scale="SLIT"
```

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

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

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

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

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

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

```
leadingPrecision="PLIT"
```

ii) Case:

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

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

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

```
scale="SLIT"
```

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

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

iii) Case:

1) If *SQLT* is INTERVAL YEAR then:

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

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

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

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

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

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

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

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

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

3) If *SQLT* is INTERVAL MONTH then:

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

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

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

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

4) If SQLT is INTERVAL DAY then:

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

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

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

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

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

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

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

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

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

6) If SQLT is INTERVAL DAY TO MINUTE then:

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

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

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

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

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

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

```
<xsd:pattern value=
  "-?P\p{Nd}{PLIT}DT\p{Nd}{2}H\p{Nd}{2}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>
  <xsd:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                      ANNT ANNP ANNS/>
```

```
  </xsd:appinfo>
</xsd:annotation>
```

v) Case:

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

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

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

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

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

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

ii) Let ***XMLN*** be the XML Name obtained by applying Subclause 9.9, "Mapping an SQL data type to an XML Name", to *DT*.

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

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

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

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

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

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

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

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

ii) Let ***XMLMT***$_i$ be the result of applying the mapping in Subclause 9.9, "Mapping an SQL data type to an XML Name", to FT_i, for *i* between 1 (one) and *N*.

iii) Let ***XMLFN$_i$*** be the result of applying the mapping in Subclause 9.1, “Mapping SQL <identifier>s to XML Names”, to *FN$_i$*, for *i* between 1 (one) and *N*, using the fully escaped variant of the mapping.

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

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

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

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

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

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

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

ii) Let ***XMLN*** be the XML Name obtained by applying Subclause 9.9, “Mapping an SQL data type to an XML Name”, to *ST*.

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

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

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

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

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

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

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

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

ii) Let ***XMLN*** be the XML Name obtained by applying Subclause 9.9, "Mapping an SQL data type to an XML Name", to *ET*.

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

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

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

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

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

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

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

ii) Let ***XMLN*** be the XML Name obtained by applying Subclause 9.9, "Mapping an SQL data type to an XML Name", to *ET*.

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

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

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

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

r) If *SQLT* is an XML type, then:

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

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

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

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

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

Conformance Rules

None.

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

Function

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

Syntax Rules

None.

Access Rules

None.

General Rules

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

   Case:

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

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

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

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

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

4) Let ***`XMLT`*** be the XML Schema data type that is the result of applying the mapping defined in Subclause 9.15, "Mapping SQL data types to XML Schema data types", to *SQLT* using *NULLS* as the choice of whether to map null values to absent elements or to elements that are marked with **`xsi:nil="true"`** and using using *ENCODING* as the choice of whether to encode binary strings in base64 or in hex.

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

6) Let *CV* be the result of

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

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

8) Case:

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

i) Let *CS* be the character set of *SQLT*. Let ***XMLVRAW*** be the result of mapping *SQLV* to Unicode using the implementation-defined mapping of character strings of *CS* to Unicode. If any Unicode code point in ***XMLVRAW*** does not represent a valid XML character, then an exception condition is raised: *SQL/XML mapping error — invalid XML character*.

ii) Let *XMLV* be *XMLVRAW*, with each instance of "&" (U+0026) replaced by "&amp;", each instance of "<" (U+003C) replaced by "&lt;", each instance of ">" (U+003E) replaced by "&gt;", and each instance of Carriage Return (U+000D) replaced by "&#x0d;".

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

Case:

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

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

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

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

```
LOWER (CV)
```

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

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

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

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

f) If *SQLT* specifies TIME, then:

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

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

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

iv) Let *TEMP* be the result of:

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

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

g) If *SQLT* specifies TIMESTAMP, then:

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

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

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

iv) Let *TEMP* be the result of:

```
SUBSTRING (CV FROM 11 FOR 10)
```

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

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

h) If *SQLT* specifies INTERVAL, then:

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

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

iii) Let *CVA* be the result of:

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

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

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

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

vii) Case:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1) Let ***XMLFN*$_i$** be the result of applying the mapping in Subclause 9.1, "Mapping SQL <identifier>s to XML Names", to *FN*$_i$, using the fully escaped variant of the mapping.

2) Case:

A) If *FV*$_i$ is the null value and *NULLS* is absent, then let ***XMLE*$_i$** be the empty string.

B) If *FV*$_i$ is the null value and *NULLS* is nil, then let ***XMLE*$_i$** be the XML element:

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

C) Otherwise, let the XML text ***XMLV*$_i$** be the result of applying the mapping defined in this Subclause to *FV*$_i$. Let ***XMLE*$_i$** be the XML element:

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

3) Let ***XMLV*** be:

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

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

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

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

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

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

2) Case:

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

B) Otherwise, let ***X*$_i$** be the result of applying this Subclause to *E*$_i$. Let the XML text ***XMLE*$_i$** be:

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

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

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

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

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

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

Conformance Rules

None.

9.17 Mapping XML Names to SQL <identifier>s

Function

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

Syntax Rules

None.

Access Rules

None.

General Rules

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

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

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

   Case:

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

      Case:

      i) If the Unicode code point U+$\boldsymbol{X}_{\mathrm{i+2}}\boldsymbol{X}_{\mathrm{i+3}}\boldsymbol{X}_{\mathrm{i+4}}\boldsymbol{X}_{\mathrm{i+5}}$ 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.

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

      Case:

      i) If the Unicode code point U+$\boldsymbol{X}_{\boldsymbol{i+2}}\boldsymbol{X}_{\boldsymbol{i+3}}\boldsymbol{X}_{\boldsymbol{i+4}}\boldsymbol{X}_{\boldsymbol{i+5}}\boldsymbol{X}_{\boldsymbol{i+6}}\boldsymbol{X}_{\boldsymbol{i+7}}$ 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}, and U_{i+8} 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}, and U_{i+8} 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) $\le i \le N$, in order by *i*.

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

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

Conformance Rules

None.

10 Additional common rules

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

10.1 Type precedence list determination

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

Function

Determine the type precedence list of a given type.

Syntax Rules

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

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.2 Type name determination

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

Function

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

Syntax Rules

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

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.3 Determination of identical values

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

Function

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

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 2)c)ii) If *V1* and *V2* are both of the XML type, then whether *V1* and *V2* are identical or not identical is defined as follows (note that the definition involves the simultaneous recursive definition of identical SQL/XML information items and identical properties of SQL/XML information items, as well as identical XML values):

   a) Two SQL/XML information items *XII1* and *XII2* are defined to be identical if they satisfy the following conditions:

      i) *XII1* and *XII2* are the same kind of SQL/XML information item (both are an XML root information item, an XML element information item, an XML attribute information item, an XML processing instruction information item, an XML unexpanded entity reference information item, an XML character information item, an XML comment information item, an XML document type declaration information item, an XML unparsed entity information item, an XML notation information item, or an XML namespace information item).

      ii) Every significant property *P* of *XII1* is identical to the corresponding property *P* of *XII2*. The significant and insignifcant properties of SQL/XML information items are shown in Table 2, “SQL/XML Information Item Properties”.

Table 2 — SQL/XML Information Item Properties

SQL/XML Information Item	Significant Properties	Insignificant Properties
root	[children] [notations] [unparsed entities] [standalone] [version]	
element	[namespace name] [local name] [children] [attributes]	[prefix] [namespace attributes] [in-scope namespaces] [base URI] [parent]
attribute	[namespace name] [local name] [normalized value] [attribute type]	[prefix] [references] [owner element]
processing instruction	[target] [content] [notation]	[parent] [base URI]
unexpanded entity reference	[name] [system identifier] [public identifier]	[declaration base URI] [parent]
character	[character code]	[parent] [element content whitespace]
comment	[content]	[parent]
document type declaration	[system identifier] [public identifier] [children]	[parent]
unparsed entity	[name] [system identifier] [public identifier] [notation name] [notation]	[declaration base URI]
notation	[name] [system identifier] [public identifier]	[declaration base URI]

SQL/XML Information Item	Significant Properties	Insignificant Properties
namespace	[namespace name]	[prefix]

b) A property *P* of an SQL/XML information item *XII1* is identical to the same property *P* of another XML information item *XII2* if

   Case:

   i) If *P* is an ordered list of XML information items, then the number of elements of property *P* in *XII1* equals the number of element of property *P* in *XII2*, and corresponding XML information items in the two lists are identical.

   ii) If *P* is a set of XML information items, then the number of elements of property *P* in *XII1* equals the number of element of property *P* in *XII2*, and there exists a one-to-one mapping of the elements of property *P* in *XII1* to the elements of property *P* in *XII2* such that corresponding XML information items in the two sets are identical.

   iii) If *P* consists of a single XML information item, then the XML information item element that is property *P* in *XII1* is identical to the XML information item that is property *P* in *XII2*.

   iv) If *P* is the [version] property of the XML root information item, then the values “no value” and “1.0” are regarded as identical; otherwise, two values are identical if they are equal.

   v) Otherwise, property *P* of *XII1* equals property *P* of *XII2*.

c) Two non-null XML values *V1* and *V2* are identical if their XML root information items are identical.

Conformance Rules

No additional Conformance Rules.

10.4 Equality operations

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

Function

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

Syntax Rules

1) $\boxed{\text{Insert this SR}}$ The declared type of an operand of an equality operation shall not be XML-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.5 Grouping operations

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

Function

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

Syntax Rules

1) $\boxed{\text{Insert this SR}}$ The declared type of an operand of a grouping operation shall not be XML-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.6 Multiset element grouping operations

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

Function

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

Syntax Rules

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

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.7 Ordering operations

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

Function

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

Syntax Rules

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

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.8 Determination of [namespace name] property

Function

Determine the XML namespace URI of an XML QName.

Syntax Rules

1) Let *QN* be the *QNAME* in an application of this Subclause, and let *B* be the *BNFTERM* in an application of this Subclause.

2) Case:

   a) If *QN* has an XML QName prefix, then:

      i) Let *P* be the XML QName prefix of *QN*.

      ii) *B* shall be within the scope of one or more <XML namespace declaration>s that contain an <XML namespace prefix> equivalent to *P*.

      iii) Let *XND* be the <XML namespace declaration> that contains an <XML namespace prefix> equivalent to *P* with innermost scope that includes *B*.

      iv) Let *XNDI* be the <XML namespace declaration item> of *XND* that contains an <XML namespace prefix> equivalent to *P*.

      v) Let *NSURI* be the <XML namespace URI> contained in *XNDI*.

   b) Otherwise:

      i) If *B* is contained within the scope of one or more <XML namespace declaration>s that contain an <XML default namespace declaration item>, then:

         1) Let *XDNDI* be the <XML default namespace declaration item> with innermost scope that includes *B*.

         2) Case:

            A) If *XDNDI* contains an <XML namespace URI>, then let *NSURI* be that <XML namespace URI>.

            B) Otherwise, let *NSURI* be the zero-length string.

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

3) Let *CS* be the character set of the declared type of *NSURI*. Let *CSM* be the implementation-defined mapping of strings of *CS* to strings of Unicode. Let *R* be the result of mapping *NSURI* to a string of Unicode using *CSM*.

4) *R* is the result of the Syntax Rules of this Subclause.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

10.9 Determination of [namespace attributes] property

Function

Determine the [namespace attributes] property of an XML element information item.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *EII* be the XML element information item *INFOITEM* in an application of this Subclause.

2) Let *P* be

   Case:

   a) If the [prefix] property of *EII* is “no value”, then the zero-length string.

   b) Otherwise, the [prefix] property of *EII*.

3) Let *SQ* be the set consisting of *P* and, for each XML attribute information item *AII* contained in the [attributes] property of *EII*, if the [prefix] property of *AII* is not “no value”, then the [prefix] property of *AII*.

   NOTE 24 — As a set, duplicates are removed from *SQ*.

4) Let *N* be the number of elements of *SQ*. Let M_i, 1 (one) ≤ *i* ≤ *N*, be an enumeration of the members of *SQ*.

5) For each *i* between 1 (one) and *N*, let $NAII_i$ be an XML attribute information item, constructed as follows:

   a) The [namespace name] property of $NAII_i$ is “`http://www.w3.org/2000/xmlns`”.

   b) The [local name] property of $NAII_i$ is

      Case:

      i) If M_i is the zero-length string, then “xmlns”.

      ii) Otherwise, M_i.

   c) The [prefix] property of $NAII_i$ is

      Case:

i) If M_i is the zero-length string, then “no value”.

ii) Otherwise, “xmlns”.

d) The [normalized value] property of $NAII_i$ is

Case:

i) If M_i is the zero-length string, then the [namespace URI] property of *EII*.

ii) Otherwise,

Case:

1) If M_i is equivalent to P, then the [namespace URI] property of *EII*.

2) Otherwise, the [namespace URI] property of any XML attribute information item contained in the [attributes] property of *EII* whose [prefix] property is equivalent to M_i.

e) The [specified] property indicates that the namespace attribute was specified.

f) The [attribute type] property is “CDATA”.

g) The [references] property is “no value”.

h) The [owner element] property is *EII*.

6) The [namespace attributes] property of *EII* is the set of XML attribute information items $NAII_i$, 1 (one) ≤ $i \leq N$.

Conformance Rules

None.

10.10 Determination of the [in-scope namespaces] property

Function

Determine the [in-scope namespaces] property of an XML element information item and its descendent XML element information items.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *EII* be the XML element information item *INFOITEM* in an application of this Subclause.

2) Case:

   a) If the [parent] property of *EII* is an XML element information item *PARENT*, then let *R* be the value of the [in-scope namespaces] property of *PARENT*.

   b) Otherwise, let *R* be the empty set.

3) Let *NA* be the [namespace attributes] property of *EII*.

4) Let *N* be the cardinality of *NA*. Let A_i, 1 (one) ≤ i ≤ N, be an enumeration of the XML attribute information items that are members of *NA*.

5) For all i between 1 (one) and *N*, let LN_i be the [local name] property of A_i and let P_i be the [prefix] property of A_i.

6) For i between 1 (one) and *N*,

   Case:

   a) If P_i is "xmlns", and there is an XML namespace information item *NSI* in *R* such that the [prefix] property of *NSI* equals LN_i, then set the [namespace name] property of *NSI* to the value of the [normalized value] property of A_i.

   b) If the value of P_i is "no value" and LN_i is "xmlns", and there is an XML namespace information item *NSI* in *R* such that the [prefix] property of *NSI* has no value, then set the [namespace name] property of *NSI* to the value of the [normalized value] property of A_i.

   c) Otherwise, insert into *R* an XML namespace information item *NSI* with the following properties:

      i) Case:

1) If P_i is “xmlns”, then the [prefix] property of *NSI* has no value.

2) Otherwise, the [prefix] property of *NSI* is LN_i.

ii) The [namespace name] property of *NSI* is the [normalized value] property of A_i.

7) If there is no XML namespace information item in *R* whose [prefix] property is “xml”, then an XML namespace information item *NSI* with the following properties is inserted into *R*:

a) The [prefix] property of *NSI* is “xml”.

b) The [namespace name] property of *NSI* is “`http://www.w3.org/XML/1998/namespace`”.

8) For every XML element information item *CHILD* contained in the [children] property of *EII*, the General Rules of this Subclause are executed recursively, with *CHILD* as the *INFOITEM* in the recursive execution.

Conformance Rules

None.

10.11 Determination of [whitespace element content] property

Function

Determine the [whitespace element content] property of an XML element information item.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *EI* be the *INFOITEM* in an application of this Subclause. The [children] property of *EI* is a list of XML information items. Let *N* be the cardinality of the [children] property of *EI*. Let XII_i, 1 (one) ≤ *i* ≤ *N*, be the list of XML information items that is the [children] property of *EI*.

2) For every *i* between 1 (one) and *N*, if XII_i is an XML character information item, then:

   a) Let *j* be the least subscript less than *i* such that for all *q* between *j* and *i*, XII_q is an XML character information item.

   b) Let *k* be the greatest subscript greater than *i* such that for all *q* between *i* and *k*, XII_q is an XML character information item.

   NOTE 25 — Thus the list XII_j , ... , XII_k is the maximal sublist of *XII* containing XII_i and consisting entirely of XML character information items. Such a maximal list of XML character information items is commonly called a “text node”.

   c) If for all *q* between *j* and *k*, XII_q is an XML character information item whose [character code] property is a whitespace character, then for all *q* between *j* and *k*, the [element content whitespace] property of XII_q is set to “true”; otherwise, for all *q* between *j* and *k*, the [element content whitespace] property of XII_q is set to “false”.

Conformance Rules

None.

10.12 Concatenation of two XML values

Function

Specify the result of the concatenation of two XML values.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *X* and *Y* be the two XML values in an application of this Subclause.

2) The result *C* of the concatenation of *X* and *Y* is an XML value defined as follows:

   Case:

   a) If *X* is the null value, then *C* is *Y*.

   b) If *Y* is the null value, then *C* is *X*.

   c) Otherwise:

      i) Let *XC* be a variable whose value is identical to *X*, and let *YC* be a variable whose value is identical to *Y*.

      ii) Let *XR* be the XML root information item of *XC*, and let *YR* be the XML root information item of *YC*.

      iii) Let *C* be a collection of SQL/XML information items, consisting of all SQL/XML information items of *XC* except *XR* and the XML document type declaration information item (if any), and all SQL/XML information items of *YC* except *YR* and the XML document type declaration information item (if any), and one XML root information item *CR* defined as follows:

         1) The [children] property of *CR* consists of the list of children of *XR*, excluding the XML document type declaration information item, if any, followed by the list of children of *YR*, excluding the XML document type declaration information item, if any.

         2) The [notations] property of *CR* consists of the set union of the [notations] property of *XR* and the [notations] property of *YR*. In forming this set union, two XML notation information items are regarded as equal if they are identical, according to the rules of Subclause 10.3, “Determination of identical values”. If two XML notation information items are identical, it is implementation-dependent which one is retained in the set union. Any discarded XML notation information items are also removed from *C*. If there are two non-identical XML

notation information items with the identical [name] property, an exception condition is raised: *data exception — non-identical notations with the same name*.

3) The [unparsed entities] property of *CR* consists of the set union of the [unparsed entities] property of *XR* and the [unparsed entities] property of *YR*. In forming this set union, two XML unparsed entity information items are regarded as equal if they are identical, according to the rules of Subclause 10.3, "Determination of identical values". If two XML unparsed entity information items are identical, it is implementation-dependent which one is retained in the set union. Any discarded XML unparsed entity information items are also removed from *C*. If there are two non-identical XML unparsed entity information items with the identical [name] property, an exception condition is raised: *data exception — nonidentical unparsed entities with the same name*.

4) The [standalone] property of *CR* is defined as follows.

   Case:

   A) If the [standalone] property of either *XR* or *YR* is "no", then "no".

   B) If the [standalone] property of either *XR* or *YR* is "no value", then "no value".

   C) Otherwise, "yes".

5) The [version] property of *CR* is defined as follows.

   Case:

   A) If the [version] property of *XR* and the [version] property of *YR* are identical, then the [version] property of *XR*.

   B) Otherwise, "no value".

iv) For each SQL/XML information item *II* in *C* that is a child of either *XR* or *YR*, the [parent] property of *II* is changed to *CR*.

3) *C* is the result of the concatenation of *X* and *Y*.

Conformance Rules

None.

10.13 Serialization of an XML value

Function

Specify the serialization of an XML value.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *V* be the XML value *VALUE* in an application of this Subclause. Let *DC* be the *SYNTAX* in an application of this Subclause (*i.e.*, either DOCUMENT or CONTENT). Let *DT* be the character string data type *TYPE* in an application of this Subclause. Let *CS* be the character set of *DT*.

2) Case:

   a) If *V* is the null value, then the result is the null value.

   b) Otherwise:

      i) Let *RII* be the XML root information item of *V*.

      ii) Case:

         1) If *DC* is DOCUMENT, then:

            A) If the [children] property of *RII* does not contain exactly one XML element information item, then an exception condition is raised: *data exception — not an XML document*.

            B) Case:

               I) If *CS* is UTF8 or UTF16, then let *CEC* be the zero-length string.

               II) If *CS* is UTF32, then let *CEC* be

               ```
               encoding="UTF-32"
               ```

               III) Otherwise, let *EN* be the implementation-defined name of the character encoding of *CS*. Let *CEC* be

               ```
               encoding="EN"
               ```

            C) Case:

               I) If the [standalone] property of *RII* is “yes”, then let *SA* be

```
standalone="yes"
```

II) If the [standalone] property of *RII* is “no”, then let *SA* be

```
standalone="no"
```

III) Otherwise, let *SA* be the zero-length string.

D) Let *VP* be the [version] property of *RII*.

Case:

I) If *VP* is not “no value”, then let *VA* be

```
version="VP"
```

II) If either *CEC* or *SA* is not the zero-length string, then let *VA* be

```
version="1.0"
```

III) Otherwise, let *VA* be the zero-length string.

E) If *VA* is not the zero-length string, then let *CV1* be

```
CAST ('<?xml VA CEC SA>' AS DT)
```

2) Otherwise, let *CV1* be the zero-length string.

iii) Let *CVTEMP* be an implementation-dependent character string value of character set UTF16 such that the result of

```
XMLPARSE (DC CVTEMP PRESERVE WHITESPACE)
```

is identical to *V*, except possibly for the [version] and [standalone] properties of the XML root information item.

NOTE 26 — [Canonical XML] is an example of an algorithm that may be used to compute *CVTEMP*.

NOTE 27 — Whether certain characters, (such as “<”) are handled by means of entity references (such as &lt;) or by use of CDATA sections, is implementation-dependent.

iv) Let *CV2* be a character string value of type *DT* such that the implementation-defined mapping from strings of *DT* to strings of Unicode, the latter expressed in UTF16, is *CVTEMP*. If there is no such value *CV2*, then an exception condition is raised: *data exception — character not in repertoire*.

v) The serialization of *V* is the result of

```
CV1 || CV2
```

Conformance Rules

None.

10.14 Parsing a character string as an XML value

Function

Parse a character string to obtain an XML value.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *DC* be the *SYNTAX* (either DOCUMENT or CONTENT), let *V* be the value of the character string *TEXT*, and let *WO* be the value of the <XML whitespace option> *OPTION* in an application of this Subclause.

2) A character string is called a *textual XML document* if the character string conforms to the definition of a well-formed XML document as defined in [XML 1.0], as modified by [Namespaces].

3) A character string is called a *textual XML content* if the following are all true:

    a) The character string matches the following production as an extension to the grammar of [XML 1.0], as modified by [Namespaces]:

    ```
    XMLvalue ::= XMLDecl? content
    ```

    b) The character string meets all the well-formedness constraints of applicable productions of [XML 1.0], as modified by [Namespaces].

    c) Each of the parsed entities, as defined by [XML 1.0], that is referenced directly or indirectly within the textual XML content is well-formed, as defined by [XML 1.0].

4) Case:

    a) If *DC* is DOCUMENT and *V* is not a textual XML document, then an exception condition is raised: *data exception — invalid XML document*.

    b) If *DC* is CONTENT and *V* is not a textual XML content, then an exception condition is raised: *data exception — invalid XML content*.

5) *V* is parsed and a collection *C* of SQL/XML information items is produced according to the rules of [Infoset], with the following modifications:

    a) Instead of an XML document information item, an XML root information item is produced, as follows:

        i) The [children] property is an ordered list of XML information items.

Case:

1) If *DC* is DOCUMENT, then the [children] property of the XML root information item is the same as would have been produced in an XML document information item, using the rules of [Infoset].

2) If *DC* is CONTENT, then the [children] property consists of the list of XML information items that would be produced using the rules of [Infoset] corresponding to the BNF nonterminal content defined in rule [43] of [XML 1.0].

ii) The [notations] property is implementation-defined.

iii) The [unparsed entities] property is implementation-defined.

iv) The [standalone] property is “yes” if the XMLDecl includes the attribute **`standalone='yes'`**, “no” if the XMLDecl includes the attribute **`standalone='no'`**, and “no value” otherwise.

v) The [version] property is the value of the version attribute of the XMLDecl, if any; otherwise, it is “no value”.

b) The [base URI] property of any XML information item is implementation-dependent.

c) References to entities that are defined in an internal DTD are replaced by their expanded form.

d) Default values defined by an internal DTD are applied.

e) Support for an external DTD is implementation-defined.

6) If *WO* is STRIP WHITESPACE, then any XML character information item *CII* contained in *C* whose [element content whitespace] property is “true” and that is contained in an XML element information item *EII1* that does not have an attribute **`xml:space='preserve'`** without an intervening XML element information item *EII2* that has an attribute **`xml:space='default'`** is removed from *C* and from the [children] property of the XML element information item that contains *CII*.

7) *C* is the result of parsing *V* as an XML value.

Conformance Rules

None.

11 Additional common elements

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

11.1 <routine invocation>

This Subclause modifies Subclause 10.4, “<routine invocation>”, in ISO/IEC 9075-2.

Function

Invoke an SQL-invoked routine.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 4)b)i)1) If T_i is the XML type, then:

   a) Let XDT_i be the associated string type of P_i.

   b) Let XO_i be the associated XML option of P_i.

   c) Case:

      i) If the character set of XDT_i is UTF16, then let BOM_i be `U&'\FEFF'`.

      ii) Otherwise, let BOM_i be the zero-length string of type XDT_i.

   d) CPV_i is replaced by the result of

      BOM_i `|| XMLSERIALIZE (`XO_i CPV_i `AS` XDT_i`)`

2) $\boxed{\text{Insert after GR 4)b)ii)1)}}$ If T_i is the XML type, then:

   a) Let XDT_i be the associated string type of P_i.

   b) CPV_i is replaced by an implementation-defined value of type XDT_i.

3) $\boxed{\text{Insert before GR 8)h)i)4)B)II)2)b)}}$ If *CRT* is the XML type, then:

   a) Let *XO* be the associated XML option of the result of *R*.

   b) Let *RDI* be the result of

```
XMLPARSE (XO ERDI STRIP WHITESPACE)
```

4) $\boxed{\text{Insert before GR 8)i)iii)2)C)}}$ If T_i is the XML type, then:

   a) Let XO_i be the associated XML option of P_i.

   b) CPV_i is set to the result of

```
XMLPARSE (XOi EVi STRIP WHITESPACE)
```

Conformance Rules

No additional Conformance Rules.

11.2 <aggregate function>

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

Function

Specify a value computed from a collection of rows.

Format

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

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

Syntax Rules

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

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 1)b) If the most specific type of the result of *AF* is XML, then an exception condition is raised: *data exception — XML value overflow*.

2) Insert this GR If <XML aggregate> is specified, then:

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

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

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

   d) Let *TXA* be ordered according to the values of the <sort key>s found in the second through (*K*+1)-th columns of *TXA*. If *K* is 0 (zero), then the ordering of *TXA* is implementation-dependent. Let *N* be the number of rows in *TXA*. Let R_i, 1 (one) ≤ *i* ≤ *N*, be the rows of *TXA* according to the ordering of *TXA*.

   e) Case:

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

ii) Otherwise:

1) Let V_1 be the value of the first column of R_1.

2) Let V_i, $2 \leq i \leq N$, be the concatenation of V_{i-1} and the value of the first column of R_i, according to the General Rules of Subclause 10.12, “Concatenation of two XML values”.

3) The result is *VN*.

Conformance Rules

1) $\boxed{\text{Insert this CR}}$ Without Feature X034, “XMLAgg”, conforming SQL language shall not contain an <XML aggregate>.

2) $\boxed{\text{Insert this CR}}$ Without Feature X035, “XMLAgg: ORDER BY option”, conforming SQL language shall not contain an <XML aggregate> that contains a <sort specification list>.

11.3 <XML query options>

Function

Declare one or more XML namespaces and the encoding to use for binary strings.

Format

```
<XML query options> ::=
    <XML query option> [ <comma> <XML query option> ]

<XML query option> ::=
    <XML namespace declaration>
  | <XML binary encoding>

<XML namespace declaration> ::=
    XMLNAMESPACES <left paren> <XML namespace declaration item>
    [ { <comma> <XML namespace declaration item> }... ] <right paren>

<XML namespace declaration item> ::=
    <XML namespace URI> AS <XML namespace prefix>
  | <XML default namespace declaration item>

<XML namespace prefix> ::= <identifier>

<XML namespace URI> ::= <character string literal>

<XML default namespace declaration item> ::=
    DEFAULT <XML namespace URI>
  | NO DEFAULT

<XML binary encoding> ::=
    XMLBINARY [ USING ] { BASE64 | HEX }
```

Syntax Rules

1) An <XML query options> shall contain at most one <XML namespace declaration>, and at most one <XML binary encoding>.
2) <XML namespace declaration> shall contain at most one <XML default namespace declaration item>.
3) Each <XML namespace prefix> shall be an XML NCName.
4) No two <XML namespace prefix>es shall be equivalent.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

12 Schema definition and manipulation

This Clause modifies Clause 11, “Schema definition and manipulation”, in ISO/IEC 9075-2.

12.1 <column definition>

This Subclause modifies Subclause 11.4, “<column definition>”, in ISO/IEC 9075-2.

Function

Define a column of a base table.

Format

```
<generation expression> ::=
    <left paren> [ WITH <XML query options> ]
    <value expression> <right paren>
```

Syntax Rules

1) Insert this SR The scope of each <XML namespace declaration item> contained in the <XML query options> is the <generation expression>.

2) Insert this SR The scope of an <XML binary encoding> contained in the <XML query options> is the <generation expression>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature X083, “XML namespace declarations in DDL”, in conforming SQL language, a <generation expression> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

2) Without Feature X133, “XMLBINARY clause in DDL”, in conforming SQL language, a <generation expression> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

3) $\boxed{\text{Insert this CR}}$ Without Feature X016, “Persistent XML values”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML type or a distinct type whose source type is the XML type.

12.2 <check constraint definition>

This Subclause modifies Subclause 11.9, "<check constraint definition>", in ISO/IEC 9075-2.

Function

Specify a condition for the SQL-data.

Format

```
<check constraint definition> ::=
    CHECK <left paren> [ WITH <XML query options> ]
    <search condition> <right paren>
```

Syntax Rules

1) $\boxed{\text{Insert this SR}}$ The scope of each <XML namespace declaration item> contained in the <XML namespace declaration> is the <check constraint definition>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) $\boxed{\text{Insert this CR}}$ Without Feature X083, "XML namespace declarations in DDL", in conforming SQL language, a <check constraint definition> shall not immediately contain and <XML query options> that contains an <XML namespace declaration>.
2) Without Feature X133, "XMLBINARY clause in DDL", in conforming SQL language, a <check constraint definition> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

12.3 <view definition>

This Subclause modifies Subclause 11.22, "<view definition>", in ISO/IEC 9075-2.

Function

Define a viewed table.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature X016, "Persistent XML values", conforming SQL language shall not contain a <view definition> that defines a column whose declared type is based on either the XML type or a distinct type whose source type is the XML type.

12.4 <assertion definition>

This Subclause modifies Subclause 11.37, "<assertion definition>", in ISO/IEC 9075-2.

Function

Specify an integrity constraint.

Format

```
<assertion definition> ::=
    CREATE ASSERTION <constraint name> CHECK
    <left paren> [ WITH <XML query options> ] <search condition> <right paren>
```

Syntax Rules

1) Insert this SR The scope of each <XML namespace declaration item> contained in the <XML query options> is the <assertion definition>.

2) Insert this SR The scope of an <XML binary encoding> contained in the <XML query options> is the <assertion definition>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature X083, "XML namespace declarations in DDL", in conforming SQL language, an <assertion definition> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

2) Without Feature X133, "XMLBINARY clause in DDL", in conforming SQL language, an <assertion definition> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

12.5 <user-defined type definition>

This Subclause modifies Subclause 11.41, "<user-defined type definition>", in ISO/IEC 9075-2.

Function

Define a user-defined type.

Format

`No additional Format items.`

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) $\boxed{\text{Insert this CR}}$ Without Feature X013, "Distinct types on XML type", conforming SQL language shall not contain a <representation> that is a <predefined type> that is XML.

12.6 <attribute definition>

This Subclause modifies Subclause 11.42, “<attribute definition>”, in ISO/IEC 9075-2.

Function

Define an attribute of a structured type.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature X014, “Attributes of XML type”, conforming SQL language shall not contain an <attribute definition> that contains a <data type> that is based on either the XML type or a distinct type whose source type is the XML type.

12.7 <SQL-invoked routine>

This Subclause modifies Subclause 11.50, "<SQL-invoked routine>", in ISO/IEC 9075-2.

Function

Define an SQL-invoked routine.

Format

```
<parameter type> ::=
    <data type> [ <locator indication> ]
    [ <document or content> ] [ <string type option> ]

<returns data type> ::=
    <data type> [ <locator indication> ]
    [ <document or content> ] [ <string type option> ]

<string type option> ::= AS <character string type>
```

Syntax Rules

1) Insert before SR 6)y) Case:

   a) If the <data type> immediately contained in a <parameter type> or <returns data type> is XML and *R* is an external routine, then:

      i) <locator indication> shall not be specified.

      ii) <document or content> and <string type option> shall be specified.

   b) Otherwise, <document or content> and <string type option> shall not be specified.

2) Insert before SR 6)y) If the <data type> immediately contained in a <returns data type> that is immediately contained in a <returns type> *RT* is the XML type, then *RT* shall not contain a <result cast>.

3) Insert after SR 20)d)iii)1)A) If the <parameter type> T_i simply contained in the *i*-th <SQL parameter declaration> P_i is the XML type, then:

   a) Let *XDT* be the data type identified by the <character string type> contained in <string type option> immediately contained in T_i. *XDT* is the *associated string type* of P_i.

   b) Let *XO* be the <document or content> immediately contained in T_i. *XO* is the *associated XML option* of P_i.

   c) The *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by *XDT*.

4) Insert after SR 20)d)iii)2)A)II)1) If *RT* is the XML type, then:

a) Let *XDT* be the data type identified by the <character string type> contained in <string type option> immediately contained in *RT*. *XDT* is the *associated string type* of the result of *R*.

b) Let *XO* be the <document or content> immediately contained in *RT*. *XO* is the *associated XML option* of the result of *R*.

c) *PT* is *XDT*.

5) Insert before SR 20)d)iii)2)B)II)2) If RFT_{i-PN} is the XML type, then:

a) Let *XDT* be the data type identified by the <character string type> contained in <string type option> immediately contained in RFT_{i-PN}. *XDT* is the associated string type of the result of *R*.

b) Let *XO* be the <document or content> immediately contained in RFT_{i-PN}. *XO* is the associated XML option of the result of *R*.

c) PT_i is *XDT*.

6) Insert after SR 20)d)iv)1)A) If the <parameter type> T_i simply contained in the *i*-th <SQL parameter declaration> is the XML type, then:

a) Let *XDT* be the data type identified by the <character string type> contained in <string type option> immediately contained in T_i. *XDT* is the *associated string type* of T_i.

b) Let *XO* be the <document or content> immediately contained in T_i. *XO* is the *associated XML option* of T_i.

c) The *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by *XDT*.

7) Insert after SR 20)e)i) If the <parameter type> T_i simply contained in the *i*-th <SQL parameter declaration> is the XML type, then:

a) Let *XDT* be the data type identified by the <character string type> contained in <string type option> immediately contained in T_i. *XDT* is the *associated string type* of T_i.

b) Let *XO* be the <document or content> immediately contained in T_i. *XO* is the *associated XML option* of T_i.

c) The *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by *XDT*.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 3)q) For every SQL parameter that has an associated string type, the routine descriptor includes the character string type descriptor of the associated string type.

2) Insert after GR 3)q) For every SQL parameter that has an associated XML option, the routine descriptor includes an indication of the associated XML option.

Conformance Rules

1) Insert this CR Without Feature X120, “XML parameters in SQL routines”, conforming SQL language shall not contain an <SQL-invoked routine> that simply contains a <language clause> that contains SQL and that simply contains a <parameter type> or a <returns data type> that contains a <data type> that is based on either the XML type or a distinct type whose source type is the XML type.

2) Insert this CR Without Feature X121, “XML parameters in external routines”, conforming SQL language shall not contain an <SQL-invoked routine> that simply contains a <language clause> that contains ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI and that simply contains a <parameter type> or a <returns data type> that contains a <data type> that is based on either the XML type or a distinct type whose source type is the XML type.

3) Insert this CR Without Feature X110, “Host language support for XML: VARCHAR mapping”, conforming SQL language shall not contain a <string type option> that contains CHARACTER VARYING, CHAR VARYING, or VARCHAR.

4) Insert this CR Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain a <string type option> that contains CHARACTER LARGE OBJECT, CHAR LARGE OBJECT, or CLOB.

5) Insert this CR Without Feature X100, “Host language support for XML: CONTENT option”, conforming SQL language shall not contain a <document or content> that is CONTENT.

6) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, conforming SQL language shall not contain a <document or content> that is DOCUMENT.

13 SQL-client modules

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

13.1 Calls to an <externally-invoked procedure>

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

Function

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

Syntax Rules

1) Insert into SR 2)e)

```
DATA_EXCEPTION_NONIDENTICAL_NOTATIONS_WITH_THE_SAME_NAME:
  constant SQLSTATE_TYPE := "2200J";
DATA_EXCEPTION_NONIDENTICAL_UNPARSED_ENTITIES_WITH_THE_SAME_NAME:
  constant SQLSTATE_TYPE := "2200K";
DATA_EXCEPTION_NOT_AN_XML_DOCUMENT:
  constant SQLSTATE_TYPE := "2200L";
DATA_EXCEPTION_INVALID_XML_DOCUMENT:
  constant SQLSTATE_TYPE := "2200M";
DATA_EXCEPTION_INVALID_XML_CONTENT:
  constant SQLSTATE_TYPE := "2200N";
DATA_EXCEPTION_XML_VALUE_OVERFLOW:
  constant SQLSTATE_TYPE := "2200R";
DATA_EXCEPTION_INVALID_COMMENT:
  constant SQLSTATE_TYPE := "2200S";
DATA_EXCEPTION_INVALID_PROCESSING_INSTRUCTION:
  constant SQLSTATE_TYPE := "2200T";
SQLXML_MAPPING_ERROR_NO_SUBCLASS:
  constant SQLSTATE_TYPE := "0N000";
SQLXML_MAPPING_ERROR_INVALID_XML_CHARACTER:
  constant SQLSTATE_TYPE := "0N002";
SQLXML_MAPPING_ERROR_UNMAPPABLE_XML_NAME:
  constant SQLSTATE_TYPE := "0N001";
WARNING_COLUMN_CANNOT_BE_MAPPED:
  constant SQLSTATE_TYPE := "01010";
```

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

13.2 <SQL-procedure statement>

This Subclause modifies Subclause 13.5, "<SQL procedure statement>", in ISO/IEC 9075-2.

Function

Define all of the SQL-statements that are <SQL procedure statement>s.

Format

```
<SQL session statement> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <set XML option statement>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

13.3 Data type correspondences

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

Function

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

Tables

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

Table 3 — Data type correspondences for Ada

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

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

Table 4 — Data type correspondences for C

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

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

Table 5 — Data type correspondences for COBOL

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

SQL Data Type	COBOL Data Type
XML	*None*

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

Table 6 — Data type correspondences for Fortran

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

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

Table 7 — Data type correspondences for M

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

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

Table 8 — Data type correspondences for Pascal

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

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

Table 9 — Data type correspondences for PL/I

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

Conformance Rules

No additional Conformance Rules.

14 Data manipulation

This Clause modifies Clause 14, “Data manipulation”, in ISO/IEC 9075-2.

14.1 <delete statement: searched>

This Subclause modifies Subclause 14.7, “<delete statement: searched>”, in ISO/IEC 9075-2.

Function

Delete rows of a table.

Format

```
<delete statement: searched> ::=
    DELETE [ WITH <XML query options> ] FROM <target table>
    [ WHERE <search condition> ]
```

Syntax Rules

1) Insert this SR The scope of each <XML namespace declaration item> contained in the <XML query options> is the <delete statement: searched>.

2) Insert this SR The scope of an <XML binary encoding> contained in the <XML query options> is the <delete statement: searched>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature X082, “XML namespace declarations in DML”, in conforming SQL language, a <delete statement: searched> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

2) Without Feature X132, “XMLBINARY clause in DML”, in conforming SQL language, a <delete statement: searched> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

14.2 <insert statement>

This Subclause modifies Subclause 14.8, "<insert statement>", in ISO/IEC 9075-2.

Function

Create new rows in a table.

Format

```
<insert statement> ::=
    INSERT [ WITH <XML query options> ] INTO <insertion target>
    <insert columns and source>
```

Syntax Rules

1) Insert this SR The scope of each <XML namespace declaration item> contained in <XML query options> is the <insert statement>.

2) Insert this SR The scope of an <XML binary encoding> contained in <XML query options> is the <insert statement>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature X082, "XML namespace declarations in DML", in conforming SQL language, an <insert statement> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

2) Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, an <insert statement> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

14.3 <merge statement>

This Subclause modifies Subclause 14.9, "<merge statement>", in ISO/IEC 9075-2.

Function

Conditionally update rows of a table, or insert new rows into a table, or both.

Format

```
<merge statement> ::=
    MERGE [ WITH <XML query options> ] INTO <target table>
    [ [ AS ] <correlation name> ]
    USING <table reference> ON <search condition>
    <merge operation specification>
```

Syntax Rules

1) $\boxed{\text{Insert this SR}}$ The scope of each <XML namespace declaration item> contained in <XML query options> is the <merge statement>.

2) $\boxed{\text{Insert this SR}}$ The scope of an <XML binary encoding> contained in <XML query options> is the <merge statement>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) $\boxed{\text{Insert this CR}}$ Without Feature X082, "XML namespace declarations in DML", in conforming SQL language, a <merge statement> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

2) Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, a <merge statement> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

14.4 <update statement: positioned>

This Subclause modifies Subclause 14.10, "<update statement: positioned>", in ISO/IEC 9075-2.

Function

Update a row of a table.

Format

```
<update statement: positioned> ::=
    UPDATE [ WITH <XML query options> ] <target table>
    SET <set clause list> WHERE CURRENT OF <cursor name>
```

Syntax Rules

1) Insert this SR The scope of each <XML namespace declaration item> contained in <XML query options> is the <update statement: positioned>.

2) Insert this SR The scope of an <XML binary encoding> contained in <XML query options> is the <update statement: positioned>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature X082, "XML namespace declarations in DML", in conforming SQL language, an <update statement: positioned> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

2) Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, an <update statement: positioned> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

14.5 <update statement: searched>

This Subclause modifies Subclause 14.11, "<update statement: searched>", in ISO/IEC 9075-2.

Function

Update rows of a table.

Format

```
<update statement: searched> ::=
    UPDATE [ WITH <XML query options> ] <target table>
    SET <set clause list> [ WHERE <search condition> ]
```

Syntax Rules

1) Insert this SR The scope of each <XML namespace declaration item> contained in <XML query options> is the <update statement: searched>.

2) Insert this SR The scope of an <XML binary encoding> contained in <XML query options> is the <update statement: searched>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature X082, "XML namespace declarations in DML", in conforming SQL language, an <update statement: searched> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

2) Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, an <update statement: searched> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

15 Control statements

This Clause modifies Clause 15, “Control statements”, in ISO/IEC 9075-4.

15.1 <compound statement>

This Subclause modifies Subclause 13.1, “<compound statement>”, in ISO/IEC 9075-4.

Function

Specify a statement that groups other statements together.

Format

```
<compound statement> ::=
    [ <beginning label> <colon> ]
    BEGIN [ [ NOT ] ATOMIC ]
    [ DECLARE <XML query options> <semicolon> ]
    [ <local declaration list> ]
    [ <local cursor declaration list> ]
    [ <local handler declaration list> ]
    [ <SQL statement list> ]
    END [ <ending label> ]
```

Syntax Rules

1) Insert this SR The scope of each <XML namespace declaration item> contained in <XML query options> is the <compound statement>.

2) Insert this SR The scope of an <XML binary encoding> contained in <XML query options> is the <compound statement>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) $\boxed{\text{Insert this CR}}$ Without Feature X084, “XML namespace declarations in compound statements”, in conforming SQL language, a <compound statement> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

2) Without Feature X134, “XMLBINARY clause in compound statements”, in conforming SQL language, a <compound statement> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

16 Session management

This Clause modifies Clause 18, “Session management”, in ISO/IEC 9075-2.

16.1 <set XML option statement>

Function

Set the XML option of the current SQL-session.

Format

```
<set XML option statement> ::= SET XML OPTION <document or content>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Case:
   a) If DOCUMENT is specified, then the XML option of the current SQL-session is set to DOCUMENT.
   b) Otherwise, the XML option of the current SQL-session is set to CONTENT.

Conformance Rules

1) Without Feature F761, “Session management”, conforming SQL language shall not contain a <set XML option statement>.
2) Without Feature X100, “Host language support for XML: CONTENT option”, conforming SQL language shall not contain a <set XML option statement> that contains CONTENT.
3) Without Feature X101, “Host language support for XML: DOCUMENT option”, conforming SQL language shall not contain a <set XML option statement> that contains DOCUMENT.

This page intentionally left blank.

17 Dynamic SQL

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

17.1 Description of SQL descriptor areas

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

Function

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

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

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

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

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

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

Conformance Rules

No additional Conformance Rules.

17.2 <input using clause>

This Subclause modifies Subclause 19.10, “<input using clause>”, in ISO/IEC 9075-2.

Function

Supply input values for an <SQL dynamic statement>.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 6)c)ii) If *SDT* is a character string type and *TDT* is the XML type, then:

   a) Let *XO* be the XML option of the current SQL-session.

   b) If the <XML value function>

   ```
   XMLPARSE (XO IV STRIP WHITESPACE)
   ```

   does not conform to the Syntax Rules of Subclause 6.7, “<XML value function>”, then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

   c) If the <XML value function>

   ```
   XMLPARSE (XO IV STRIP WHITESPACE)
   ```

   does not conform to the General Rules of Subclause 6.7, “<XML value function>”, then an exception condition is raised in accordance with the General Rules of Subclause 6.7, “<XML value function>”.

   d) The <XML value function>

   ```
   XMLPARSE (XO IV STRIP WHITESPACE)
   ```

   is effectively performed and is the value of the *i*-th input dynamic parameter.

Conformance Rules

No additional Conformance Rules.

17.3 <output using clause>

This Subclause modifies Subclause 19.11, “<output using clause>”, in ISO/IEC 9075-2.

Function

Supply output values for an <SQL dynamic statement>.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 6)d)ii) If *SDT* is the XML type and *TDT* is a character string type, then:

   a) Let *XO* be the XML option of the current SQL-session.

   b) Case:

      i) If the character set of *TDT* is UTF16, then let *BOM* be `U&'\FEFF'`.

      ii) Otherwise, let *BOM* be the zero-length string of type *TDT*.

   c) If the <string value function>

      ```
      XMLSERIALIZE (XO SV AS TDT)
      ```

      does not conform to the Syntax Rules of Subclause 6.5, “<string value function>”, then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

   d) If the <string value function>

      ```
      XMLSERIALIZE (XO SV AS TDT)
      ```

      does not conform to the General Rules of Subclause 6.5, “<string value function>”, then an exception condition is raised in accordance with the General Rules of Subclause 6.5, “<string value function>”

   e) The <string value expression>

      ```
      BOM || XMLSERIALIZE (XO SV AS TDT)
      ```

is effectively performed and is the value *TV* of the ***i***-th <target specification>.

Conformance Rules

No additional Conformance Rules.

18 Embedded SQL

This Clause modifies Clause 20, “Embedded SQL”, in ISO/IEC 9075-2.

18.1 <embedded SQL host program>

This Subclause modifies Subclause 20.1, “<embedded SQL host program>”, in ISO/IEC 9075-2.

Function

Specify an <embedded SQL host program>.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 21)h)i)5) If *EVN* identifies an XML VARCHAR host variable, then:
   a) Let *L* be the length of *EVN*.
   b) Let *CS* be the character set of *EVN*.
   c) *PT* is

   ```
   VARCHAR(L) CHARACTER SET CS
   ```

2) Insert after SR 21)h)i)5) If *EVN* identifies an XML CLOB host variable, then:
   a) Let *L* be the length of *EVN*.
   b) Let *CS* be the character set of *EVN*.
   c) *PT* is

   ```
   CLOB(L) CHARACTER SET CS
   ```

3) Insert before SR 21)h)ii)2) If *EVN* identifies an XML VARCHAR host variable or an XML CLOB host variable, then:
   a) Let *XO* be the XML option of *EVN*.
   b) *EVN* is replaced by

```
XMLPARSE (XO EVN STRIP WHITESPACE)
```

4) Insert after SR 21)l)i)3)B)V) If HV_i identifies an XML VARCHAR host variable, then:

   a) Let L be the length of HV_i.

   b) Let CS be the character set of HV_i.

   c) PT_i is

   ```
   VARCHAR(L) CHARACTER SET CS
   ```

5) Insert after SR 21)l)i)3)B)V) If HV_i identifies an XML CLOB host variable, then:

   a) Let L be the length of HV_i.

   b) Let CS be the character set of HV_i.

   c) PT_i is

   ```
   CLOB(L) CHARACTER SET CS
   ```

6) Insert before SR 21)l)i)7) For each HVN_i , 1 (one) $\leq i \leq n$, that identifies some HV_i that is either an XML VARCHAR host variable or an XML CLOB host variable, apply the Syntax Rules of Subclause 9.6, "Host parameter mode determination", in ISO/IEC 9075-2, with the PD_i corresponding to HVN_i and ES as <host parameter declaration> and <SQL procedure statement>, respectively, to determine whether the corresponding XP_i is an input host parameter, an output host parameter, or both an input host parameter and an output host parameter.

   a) Among XP_i, 1 (one) $\leq i \leq n$, let d be the number of input host parameters, e be the number of output host parameters, and f be the number of host parameters that are both input host parameters and output host parameters.

   b) Among XP_i, 1 (one) $\leq i \leq n$, let XPI_j, 1 (one) $\leq j \leq d$, be the input host parameters, let XPO_k, 1 (one) $\leq k \leq e$, be the output host parameters, and let $XPIO_l$, 1 (one) $\leq l$ f, be the host parameters that are both input host parameters and output host parameters.

   c) For 1 (one) $\leq j \leq d$:

      i) Let $XPNI_j$ be the <host parameter name> of XPI_j.

      ii) Let $XHVI_j$ be the host variable corresponding to XPI_j.

      iii) Let $XDOCI_j$ be the XML option of $XHVI_j$.

   d) For 1 (one) $\leq k \leq e$:

      i) Let $XPNO_k$ be the <host parameter name> of XPO_k.

ii) Let $XHVO_k$ be the host variable corresponding to XPO_k.

iii) Let $XDOCO_k$ be the XML option of $XHVO_k$.

iv) Let XLO_k be the length of $XHVO_k$.

v) Let $XCSO_k$ be the character set of $XHVO_k$.

vi) Case:

1) If $XHVO_k$ is an XML VARCHAR host variable, then let XTO_k be

```
VARCHAR(XLOk) CHARACTER SET XCSOk
```

2) Otherwise, let XTO_k be

```
CLOB(XLOk) CHARACTER SET XCSOk
```

vii) Case:

1) If $XCSO_k$ is UTF16, then let $OBOM_k$ be `U&'\FEFF'`.

2) Otherwise, let $OBOM_k$ be the zero-length string of type XTO_k.

e) For 1 (one) $\leq l \leq f$:

i) Let $XPNIO_l$ be the <host parameter name> of $XPIO_l$.

ii) Let $XHVIO_l$ be the host variable corresponding to $XPIO_l$.

iii) Let $XDOCIO_l$ be the XML option of $XHVIO_l$.

iv) Let $XLIO_l$ be the length of $XHVIO_l$.

v) Let $XCSIO_l$ be the character set of $XHVIO_l$.

vi) Case:

1) If $XHVIO_l$ is an XML VARCHAR host variable, then let $XTIO_l$ be

```
VARCHAR(XLIOl) CHARACTER SET XCSIOl
```

2) Otherwise, let $XTIO_l$ be

```
CLOB(XLIOl) CHARACTER SET XCSIOl
```

vii) Case:

1) If $XCSIO_l$ is UTF16, then let $SIOBOM_l$ be `U&'\FEFF'`.

2) Otherwise, let $IOBOM_l$ be the zero-length string of type $XTIO_l$.

f) Let XVI_j, 1 (one) ≤ j ≤ d, XVO_k, 1 (one) ≤ k ≤ e, and $XVIO_l$, 1 (one) ≤ l ≤ f, be implementation-dependent <SQL variable name>s, each of which is not equivalent to any other <SQL variable name> contained in *ES*, to any <SQL parameter name> contained in *ES*, or to any <column name> contained in *ES*.

7) Insert before SR 21)l)i)7)B) If HV_i identifies an XML VARCHAR host variable or an XML CLOB host variable, then

Case:

a) If P_i is an input host parameter, then let $PXNI_j$, 1(one) ≤ j ≤ d, be the <host parameter name> of the input host parameter that corresponds to P_i; HVN_i is replaced by XVI_j.

b) If P_i is an output host parameter, then let $PXNO_k$, 1 (one) ≤ k ≤ e, be the <host parameter name> of the output host parameter that corresponds to P_i; HVN_i is replaced by XVO_k.

c) Otherwise, let $PXNIO_l$, 1 (one) ≤ l ≤ f, be the <host parameter name> of the input host parameter and the output host parameter that corresponds to P_i; HVN_i is replaced by $XVIO_l$.

8) Replace SR 21)l)i)8) The <SQL procedure statement> of *PS* is:

```
BEGIN ATOMIC
    DECLARE SVI1 TUI1;
    ...
    DECLARE SVIa TUIa;
    DECLARE XVI1 XML;
    ...
    DECLARE XVId XML;
    DECLARE SVO1 TUO1;
    ...
    DECLARE SVOb TUOb;
    DECLARE XVO1 XML;
    ...
    DECLARE XVOe XML;
    DECLARE SVIO1 TUIO1;
    ...
    DECLARE SVIOc TUIOc;
    DECLARE XVIO1 XML;
    ...
    DECLARE XVIOf XML;
    SET SVI1 = TSIN1 (CAST (PNI1 AS TTI1));
    ...
    SET SVIa = TSINa (CAST (PNIa AS TTIa));
    SET XVI1 = XMLPARSE (XDOCI1 XPNI1 STRIP WHITESPACE);
    ...
    SET XVId = XMLPARSE (SDOCId XPNId STRIP WHITESPACE);
    SET SVIO1 = TSION1 (CAST (PNIO1 AS TTIO1));
    ...
    SET SVIOc = TSIONc (CAST (PNIOc AS TTIOc));
    SET XVIO1 = XMLPARSE (XDOCIO1 XPNIO1 STRIP WHITESPACE);
    ...
    SET XVIO_f = XMLPARSE (SDOCIO_f XPNIO_f STRIP WHITESPACE);
    NES;
    SET PNO_1 = CAST ( FSON_1 (SVO_1) AS TSO_1);
    ...
    SET PNO_b = CAST ( FSON_b (SVO_b) AS TSO_b);
    SET XPNO_1 = OBOM_1 || XMLSERIALIZE ( XDOCO_1 XVO_1 AS XTO_1 );
    ...
    SET XPNO_e = OBOM_e || XMLSERIALIZE ( XDOCO_e XVO_e AS XTO_e );
    SET PNIO_1 = CAST ( FSION_1 (SVIO_1) AS TSIO_1);
    ...
    SET PNIO_c = CAST ( FSION_c (SVIO_c) AS TSIO_c);
    SET XPNIO_1 = IOBOM_1 || XMLSERIALIZE ( XDOCIO_1 XVIO_1 AS XTIO_1 );
    ...
    SET XPNIO_f = IOBOM_f || XMLSERIALIZE ( XDOCIO_f XVIO_f AS XTIO_f );
END;
```

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

18.2 <embedded SQL Ada program>

This Subclause modifies Subclause 20.3, "<embedded SQL Ada program>", in ISO/IEC 9075.

Function

Specify an <embedded SQL Ada program>.

Format

```
<Ada derived type specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <Ada XML CLOB variable>

<Ada XML CLOB variable> ::=
    SQL TYPE IS XML <document or content>
    AS CLOB <left paren> <large object length> <right paren>
    [ CHARACTER SET [ IS ] <character set specification> ]
```

Syntax Rules

1) Insert after SR 5)c) The syntax

   ```
   SQL TYPE IS XML XO AS CLOB (L)
       CHARACTER SET IS CS
   ```

   for a given <Ada host identifier> *HVN* shall be replaced by

   ```
   TYPE HVN IS RECORD
     HVN_RESERVED : Interfaces.SQL.INT ;
     HVN_LENGTH : Interfaces.SQL.INT ;
     HVN_DATA : Interfaces.SQL.CHAR (1..L);
   END RECORD;
   ```

   in any <Ada XML CLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.1, "<token> and <separator>", *XO* is <document or content> as specified in Subclause 6.5, "<string value function>", and *CS* is a <character set name> as specified in Subclause 5.4, "Names and identifiers", in ISO/IEC 9075-2. *HVN* is an *XML CLOB host variable*. *L* is the *length of XML CLOB host variable*. *CS* is the *character set of XML CLOB host variable*. *XO* is the *XML option of XML CLOB host variable*.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) $\boxed{\text{Insert this CR}}$ Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain an <Ada XML CLOB variable>.

2) $\boxed{\text{Insert this CR}}$ Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <Ada XML CLOB variable> shall not immediately contain a <document or content> that is CONTENT.

3) $\boxed{\text{Insert this CR}}$ Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <Ada XML CLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

18.3 <embedded SQL C program>

This Subclause modifies Subclause 20.4, "<embedded SQL C program>", in ISO/IEC 9075.

Function

Specify an <embedded SQL C program>.

Format

```
<C derived variable> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <C XML VARCHAR variable>
  | <C XML CLOB variable>

<C XML VARCHAR variable> ::=
    SQL TYPE IS XML <document or content> AS VARCHAR
    [ CHARACTER SET [ IS ] <character set specification> ]
    <C host identifier> <C array specification> [ <C initial value> ]
    [ { <comma> <C host identifier> <C array specification> [ <C initial value> ] }... ]

<C XML CLOB variable> ::=
    SQL TYPE IS XML <document or content>
    AS CLOB <left paren> <large object length> <right paren> [ CHARACTER SET [ IS ]
    <character set specification> ] <C host identifier> [ <C initial value> ]
    [ { <comma> <C host identifier> [ <C initial value> ] }... ]
```

Syntax Rules

1) Replace SR 5)a) Any optional CHARACTER SET specification shall be removed from a <C VARCHAR variable>, a <C character variable>, a <C CLOB variable>, a <C NCHAR variable>, <C NCHAR VARYING variable>, a <C NCLOB variable>, a <C XML VARCHAR variable>, or a <C XML CLOB variable>.

2) Replace SR 5)c) The <length> specified in a <C array specification> in any <C character variable> whose <C character type> specifies "char" or "unsigned char", in any <C VARCHAR variable>, in any <C NCHAR variable>, in any <C NCHAR VARYING variable>, or in any <C XML VARCHAR variable>, and the <large object length> specified in a <C CLOB variable> that contains a CHARACTER SET specification, a <C NCLOB variable>, or a <C XML CLOB variable> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <C host identifier> specified in the containing <C variable definition>.

3) Insert after SR 5)f) The syntax

```
SQL TYPE IS XML XO AS VARCHAR
  CHARACTER SET IS CS hvn[L]
```

for a given <C host identifier> *hvn* shall be replaced by:

```
char hvn [L]
```

in any <C XML VARCHAR variable>, where *L* is the numeric value of <length> as specified in Subclause 5.1, “<token> and <separator>”, *XO* is <document or content> as specified in Subclause 6.5, “<string value function>”, and *CS* is a <character set name> as specified in Subclause 5.4, “Names and identifiers”, in ISO/IEC 9075-2. *hvn* is an *XML VARCHAR host variable*. *L* is the *length of XML VARCHAR host variable*. *CS* is the *character set of XML VARCHAR host variable*. *XO* is the *XML option of XML VARCHAR host variable*.

4) Insert after SR 5)f) The syntax

```
SQL TYPE IS XML XO AS CLOB(L)
  CHARACTER SET IS CS hvn
```

for a given <C host identifier> *hvn* shall be replaced by:

```
struct {
  long          hvn_reserved;
  unsigned long hvn_length;
  char          hvn_data[L];
} hvn
```

in any <C XML CLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.1, “<token> and <separator>”, *XO* is <document or content> as specified in Subclause 6.5, “<string value function>”, and *CS* is a <character set name> as specified in Subclause 5.4, “Names and identifiers”, in ISO/IEC 9075-2. *hvn* is an *XML CLOB host variable*. *L* is the *length of XML CLOB host variable*. *CS* is the *character set of XML CLOB host variable*. *XO* is the *XML option of XML CLOB host variable*.

5) Replace SR 9) In a <C variable definition>, the words “VARCHAR”, “CHARACTER”, “SET”, “IS”, “VARYING”, “BLOB”, “CLOB”, “NCHAR”, “NCLOB”, “AS”, “LOCATOR”, “REF”and “XML” may be specified in any combination of upper-case and lower-case letters (see the Syntax Rules of Subclause 5.1, “<token> and <separator>”).

6) Insert after SR 10) In a <C XML VARCHAR variable>, if a <character set specification> is specified, then the SQL data type of the character string resulting from the invocation of the XMLSERIALIZE operator on a value of XML type is VARCHAR whose character set is the same as the character set specified by the <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

7) Insert after SR 10) In a <C XML CLOB variable>, if a <character set specification> is specified, then the SQL data type of the character string resulting from the invocation of the XMLSERIALIZE operator on a value of XML type is CHARACTER LARGE OBJECT whose character set is the same as the character set specified by the <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

8) Insert after SR 10) Each <C host identifier> specified in a <C XML VARCHAR variable> describes a variable-length character string. The maximum length is specified by the <length> of the <C array specification>. The value in the host variable is terminated by a null character and the position occupied by this null character is included in the maximum length of the host variable. The SQL data type of the character string resulting from the invocation of the XMLSERIALIZE operator on a value of XML type is CHARACTER VARYING whose maximum length is 1 (one) less than the <length> of the <C array specification> and whose value does not include the terminating null character. The <length> shall be greater than 1 (one).

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature X110, “Host language support for XML: VARCHAR mapping”, conforming SQL language shall not contain an <C XML VARCHAR variable>.

2) Insert this CR Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain an <C XML CLOB variable>.

3) Insert this CR Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, neither <C XML VARCHAR variable> nor <C XML CLOB variable> shall immediately contain a <document or content> that is CONTENT.

4) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, neither <C XML VARCHAR variable> nor <C XML CLOB variable> shall immediately contain a <document or content> that is DOCUMENT.

18.4 <embedded SQL COBOL program>

This Subclause modifies Subclause 20.5, “<embedded SQL COBOL program>”, in ISO/IEC 9075.

Function

Specify an <embedded SQL COBOL program>.

Format

```
<COBOL derived type specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <COBOL XML CLOB variable>

<COBOL XML CLOB variable> ::=
    [ USAGE [ IS ] ] SQL TYPE IS XML <document or content>
    AS CLOB <left paren> <large object length> <right paren>
    [ CHARACTER SET [ IS ] <character set specification> ]
```

Syntax Rules

1) Replace SR 5)b) The <length> specified in any <COBOL character type> and the <large object length> specified in any <COBOL CLOB variable>, <COBOL NCLOB variable>, or <COBOL XML CLOB variable> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <COBOL host identifier> specified in the containing <COBOL variable definition>.

2) Insert after SR 5)c) The syntax

```
SQL TYPE IS XML XO AS CLOB ( L )
    CHARACTER SET IS CS
```

for a given <COBOL host identifier> *HVN* shall be replaced by:

```
49 HVN-RESERVED PIC S9(9) USAGE IS BINARY.
49 HVN-LENGTH PIC S9(9) USAGE IS BINARY.
49 HVN-DATA PIC X(L).
```

in any <COBOL XML CLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.1, “<token> and <separator>”, *XO* is <document or content> as specified in Subclause 6.5, “<string value function>”, and *CS* is a <character set name> as specified in Subclause 5.4, “Names and identifiers”, in ISO/IEC 9075-2. *HVN* is an *XML CLOB host variable*. *L* is the *length of XML CLOB host variable*. *CS* is the *character set of XML CLOB host variable*. *XO* is the *XML option of XML CLOB host variable*.

3) Insert after SR 8) A <COBOL XML CLOB variable> describes a character string resulting from the invocation of the XMLSERIALIZE operator on a value of XML type, whose equivalent SQL data type is CHARACTER LARGE OBJECT with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) $\boxed{\text{Insert this CR}}$ Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain an <COBOL XML CLOB variable>.

2) $\boxed{\text{Insert this CR}}$ Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <COBOL XML CLOB variable> shall not immediately contain a <document or content> that is CONTENT.

3) $\boxed{\text{Insert this CR}}$ Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <COBOL XML CLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

18.5 <embedded SQL Fortran program>

This Subclause modifies Subclause 20.6, "<embedded SQL Fortran program>", in ISO/IEC 9075.

Function

Specify an <embedded SQL Fortran program>.

Format

```
<Fortran derived type specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <Fortran XML CLOB variable>

<Fortran XML CLOB variable> ::=
    SQL TYPE IS XML <document or content>
    AS CLOB <left paren> <large object length> <right paren>
    [ CHARACTER SET [ IS ] <character set specification> ]
```

Syntax Rules

1) Replace SR 6)b) The <length> specified in the CHARACTER alternative of any <Fortran type specification> and the <large object length> specified in any <Fortran CLOB variable> or <Fortran XML CLOB variable> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <Fortran host identifier> specified in the containing <Fortran variable definition>.

2) Insert after SR 6)c) The syntax

```
SQL TYPE IS XML XO AS CLOB ( L )
  CHARACTER SET IS CS
```

for a given <Fortran host identifier> HVN shall be replaced by

```
INTEGER   HVN_RESERVED
INTEGER   HVN_LENGTH
CHARACTER HVN_DATA [ <asterisk> L ]
```

in any <Fortran XML CLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.1, "<token> and <separator>", *XO* is <document or content> as specified in Subclause 6.5, "<string value function>", and *CS* is a <character set name> as specified in Subclause 5.4, "Names and identifiers", in ISO/IEC 9075-2. *HVN* is an *XML CLOB host variable*. *L* is the *length of XML CLOB host variable*. *CS* is the *character set of XML CLOB host variable*. *XO* is the *XML option of XML CLOB host variable*.

3) Insert after SR 9) A <Fortran XML CLOB variable> describes a character string resulting from the invocation of the XMLSERIALIZE operator on a value of XML type, whose equivalent SQL data type is CHARACTER LARGE OBJECT with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain an <Fortran XML CLOB variable>.

2) Insert this CR Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <Fortran XML CLOB variable> shall not immediately contain a <document or content> that is CONTENT.

3) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <Fortran XML CLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

18.6 <embedded SQL MUMPS program>

This Subclause modifies Subclause 20.7, “<embedded SQL MUMPS program>”, in ISO/IEC 9075.

Function

Specify an <embedded SQL MUMPS program>.

Format

```
<MUMPS derived type specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <MUMPS XML CLOB variable>

<MUMPS XML CLOB variable> ::=
    SQL TYPE IS XML <document or content>
    AS CLOB <left paren> <large object length> <right paren>
    [ CHARACTER SET [ IS ] <character set specification> ]
```

Syntax Rules

1) Insert after SR 9)b) The syntax

```
SQL TYPE IS XML XO AS CLOB ( L )
  CHARACTER SET IS CS
```

for a given <MUMPS host identifier> *HVN* shall be replaced by

```
INTEGER HVN_RESERVED
INTEGER HVN_LENGTH
VARCHAR HVN_DATA L
```

in any <MUMPS XML CLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.1, “<token> and <separator>”, *XO* is <document or content> as specified in Subclause 6.5, “<string value function>”, and *CS* is a <character set name> as specified in Subclause 5.4, “Names and identifiers”, in ISO/IEC 9075-2. *HVN* is an *XML CLOB host variable*. *L* is the *length of XML CLOB host variable*. *CS* is the *character set of XML CLOB host variable*. *XO* is the *XML option of XML CLOB host variable*.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) $\boxed{\text{Insert this CR}}$ Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain a <MUMPS XML CLOB variable>.

2) $\boxed{\text{Insert this CR}}$ Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <MUMPS XML CLOB variable> shall not immediately contain a <document or content> that is CONTENT.

3) $\boxed{\text{Insert this CR}}$ Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <MUMPS XML CLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

18.7 <embedded SQL Pascal program>

This Subclause modifies Subclause 20.8, "<embedded SQL Pascal program>", in ISO/IEC 9075.

Function

Specify an <embedded SQL Pascal program>.

Format

```
<Pascal derived type specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <Pascal XML CLOB variable>

<Pascal XML CLOB variable> ::=
    SQL TYPE IS XML <document or content>
    AS CLOB <left paren> <large object length> <right paren>
    [ CHARACTER SET [ IS ] <character set specification> ]
```

Syntax Rules

1) Replace SR 5)a) Any optional CHARACTER SET specification shall be removed from the PACKED ARRAY OF CHAR or CHAR alternatives of a <Pascal type specification>, a <Pascal CLOB variable>, and a <Pascal XML CLOB variable>.

2) Replace SR 5)b) The <length> specified in the PACKED ARRAY OF CHAR alternative of any <Pascal type specification> that contains a CHARACTER SET specification and the <large object length> specified in a <Pascal CLOB variable> or a <Pascal XML CLOB variable> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <Pascal host identifier> specified in the containing <Pascal variable definition>.

3) Insert after SR 5)d) The syntax

   ```
   SQL TYPE IS XML XO AS CLOB ( L )
     CHARACTER SET IS CS
   ```

   for a given <Pascal host identifier> *HVN* shall be replaced by

   ```
   VAR HVN = RECORD
     HVN_RESERVED : INTEGER ;
     HVN_LENGTH : INTEGER;
     HVN_DATA : PACKED ARRAY [ 1..L ] OF CHAR ;
   END ;
   ```

   in any <Pascal XML CLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.1, "<token> and <separator>", *XO* is <document or content> as specified in Subclause 6.5, "<string value function>", and *CS* is a <character set name> as specified in Subclause 5.4, "Names and identifiers", in ISO/IEC 9075-2. *HVN* is an *XML CLOB host variable*. *L* is the *length of XML CLOB host variable*. *CS* is the *character set of XML CLOB host variable*. *XO* is the *XML option of XML CLOB host variable*.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) $\boxed{\text{Insert this CR}}$ Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain an <Pascal XML CLOB variable>.

2) $\boxed{\text{Insert this CR}}$ Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <Pascal XML CLOB variable> shall not immediately contain a <document or content> that is CONTENT.

3) $\boxed{\text{Insert this CR}}$ Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <Pascal XML CLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

18.8 <embedded SQL PL/I program>

This Subclause modifies Subclause 20.9, “<embedded SQL PL/I program>”, in ISO/IEC 9075.

Function

Specify an <embedded SQL PL/I program>.

Format

```
<PL/I derived type specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <PL/I XML VARCHAR variable>
  | <PL/I XML CLOB variable>

<PL/I XML VARCHAR variable> ::=
    SQL TYPE IS XML <document or content>
    AS VARCHAR <left paren> <length> <right paren>
    [ CHARACTER SET [ IS ] <character set specification> ]

<PL/I XML CLOB variable> ::=
    SQL TYPE IS XML <document or content>
    AS CLOB <left paren> <large object length> <right paren>
    [ CHARACTER SET [ IS ] <character set specification> ]
```

Syntax Rules

1) Replace SR 5)b) The <length> specified in the CHARACTER, CHARACTER VARYING, or VARCHAR alternatives of any <PL/I type specification>, the <length> specified in a <PL/I XML VARCHAR variable>, and the <large object length> specified in a <PL/I CLOB variable>, or a <PL/I XML CLOB variable> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *P*N is the <PL/I host identifier> specified in the containing <PL/I variable definition>.

2) Insert after SR 5)c) The syntax

   ```
   SQL TYPE IS XML XO AS VARCHAR (L)
     CHARACTER SET IS CS
   ```

   for a given <PL/I host identifier> *HVN* shall be replaced by:

   ```
   CHARACTER VARYING [L]
   ```

   in any <PL/I XML VARCHAR variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.1, “<token> and <separator>”, *XO* is <document or content> as specified in Subclause 6.5, “<string value function>”, and *CS* is a <character set name> as specified in Subclause 5.4, “Names and identifiers”, in ISO/IEC 9075-2. *HVN* is an *XML CLOB host variable*. *L* is the *length of XML CLOB host variable*. *CS* is the *character set of XML CLOB host variable*. *XO* is the *XML option of XML CLOB host variable*.

3) Insert after SR 5)c) The syntax

```
SQL TYPE IS XML XO AS CLOB ( L )
  CHARACTER SET IS CS
```

for a given <PL/I host identifier> *HVN* shall be replaced by:

```
DCL 1 HVN
      2 HVN_RESERVED FIXED BINARY (31),
      2 HVN_LENGTH   FIXED BINARY (31),
      2 HVN_DATA     CHARACTER (<large object length> );
```

in any <PL/I XML CLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.1, “<token> and <separator>”, *XO* is <document or content> as specified in Subclause 6.5, “<string value function>”, and *CS* is a <character set name> as specified in Subclause 5.4, “Names and identifiers”, in ISO/IEC 9075-2. *HVN* is an *XML CLOB host variable*. *L* is the *length of XML CLOB host variable*. *CS* is the *character set of XML CLOB host variable*. *XO* is the *XML option of XML CLOB host variable*.

4) Insert after SR 9) In a <PL/I XML VARCHAR variable>, if a <character set specification> is specified, then the SQL data type of the character string resulting from the invocation of the XMLSERIALIZE operator on a value of XML type is VARCHAR whose character set is the same as the character set specified by the <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

5) Insert after SR 10) In a <PL/I XML CLOB variable>, if a <character set specification> is specified, then the SQL data type of the character string resulting from the invocation of the XMLSERIALIZE operator on a value of XML type is CHARACTER LARGE OBJECT whose character set is the same as the character set specified by the <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature X110, “Host language support for XML: VARCHAR mapping”, conforming SQL language shall not contain an <PL/I XML VARCHAR variable>.

2) Insert this CR Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain an <PL/I XML CLOB variable>.

3) Insert this CR Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, neither <PL/I XML VARCHAR variable> nor <PL/I XML CLOB variable> shall immediately contain a <document or content> that is CONTENT.

4) $\boxed{\text{Insert this CR}}$ Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, neither <PL/I XML VARCHAR variable> nor <PL/I XML CLOB variable> shall immediately contain a <document or content> that is DOCUMENT.

This page intentionally left blank.

19 Diagnostics management

This Clause modifies Clause 22, "Diagnostics management", in ISO/IEC 9075-2.

19.1 <get diagnostics statement>

This Subclause modifies Subclause 22.1, "<get diagnostics statement>", in ISO/IEC 9075-2.

Function

Get exception or completion condition information from the diagnostics area.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Augment Table 31, "SQL-statement codes"

Table 11 — SQL-statement codes

SQL-statement	Identifier	Code
All alternatives from ISO/IEC 9075-2		
<set XML option statement>	SET XML OPTION	138

Conformance Rules

No additional Conformance Rules.

20 Information Schema

This Clause modifies Clause 5, “Information Schema”, in ISO/IEC 9075-11.

20.1 PARAMETERS view

This Subclause modifies Subclause 5.34, “PARAMETERS view”, in ISO/IEC 9075-11.

Function

Identify the SQL parameters of SQL-invoked routines defined in this catalog that are accessible to a given user or role.

Definition

Replace the PARAMETERS view with the following

```
CREATE VIEW PARAMETERS AS
  SELECT P1.SPECIFIC_CATALOG, P1.SPECIFIC_SCHEMA, P1.SPECIFIC_NAME,
      P1.ORDINAL_POSITION, P1.PARAMETER_MODE,
      P1.IS_RESULT, P1.AS_LOCATOR, P1.PARAMETER_NAME,
      P1.FROM_SQL_SPECIFIC_CATALOG, P1.FROM_SQL_SPECIFIC_SCHEMA,
          P1.FROM_SQL_SPECIFIC_NAME,
      P1.TO_SQL_SPECIFIC_CATALOG, P1.TO_SQL_SPECIFIC_SCHEMA, P1.TO_SQL_SPECIFIC_NAME,
      D1.DATA_TYPE, D1.CHARACTER_MAXIMUM_LENGTH, D1.CHARACTER_OCTET_LENGTH,
      D1.CHARACTER_SET_CATALOG, D1.CHARACTER_SET_SCHEMA, D1.CHARACTER_SET_NAME,
      D1.COLLATION_CATALOG, D1.COLLATION_SCHEMA, D1.COLLATION_NAME,
      D1.NUMERIC_PRECISION, D1.NUMERIC_PRECISION_RADIX, D1.NUMERIC_SCALE,
      D1.DATETIME_PRECISION, D1.INTERVAL_TYPE, D1.INTERVAL_PRECISION,
      D1.USER_DEFINED_TYPE_CATALOG AS UDT_CATALOG,
      D1.USER_DEFINED_TYPE_SCHEMA AS UDT_SCHEMA,
      D1.USER_DEFINED_TYPE_NAME AS UDT_NAME,
      D1.SCOPE_CATALOG, D1.SCOPE_SCHEMA, D1.SCOPE_NAME,
      D1.MAXIMUM_CARDINALITY, D1.DTD_IDENTIFIER,
      D2.DATA_TYPE AS XML_CHAR_TYPE,
      D2.CHARACTER_MAXIMUM_LENGTH AS XML_CHAR_MAX_LEN,
      D2.CHARACTER_OCTET_LENGTH AS XML_CHAR_OCT_LEN,
      D2.CHARACTER_SET_CATALOG AS XML_CHAR_SET_CAT,
      D2.CHARACTER_SET_SCHEMA AS XML_CHAR_SET_SCH,
      D2.CHARACTER_SET_NAME AS XML_CHAR_SET_NAME,
      D2.COLLATION_CATALOG AS XML_CHAR_COLL_CAT,
      D2.COLLATION_SCHEMA AS XML_CHAR_COLL_SCH,
      D2.COLLATION_NAME AS XML_CHAR_COLL_NAME,
      D2.DTD_IDENTIFIER AS XML_CHAR_DTD_ID,
      P1.XML_OPTION
  FROM ( DEFINITION_SCHEMA.PARAMETERS P1
       LEFT JOIN
         DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR D1
         ON ( P1.SPECIFIC_CATALOG, P1.SPECIFIC_SCHEMA, P1.SPECIFIC_NAME,
              'ROUTINE', P1.DTD_IDENTIFIER )
          = ( D1.OBJECT_CATALOG, D1.OBJECT_SCHEMA, D1.OBJECT_NAME,
              D1.OBJECT_TYPE, D1.DTD_IDENTIFIER )
       LEFT JOIN
         DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR D2
         ON ( P1.SPECIFIC_CATALOG, P1.SPECIFIC_SCHEMA, P1.SPECIFIC_NAME,
              'ROUTINE', P1.XML_STRING_DTD_IDENTIFIER )
          = ( D2.OBJECT_CATALOG, D2.OBJECT_SCHEMA, D2.OBJECT_NAME,
              D2.OBJECT_TYPE, D2.DTD_IDENTIFIER ) )
    JOIN
       DEFINITION_SCHEMA.ROUTINES R1
      ON ( ( P1.SPECIFIC_CATALOG, P1.SPECIFIC_SCHEMA, P1.SPECIFIC_NAME )
         = ( R1.SPECIFIC_CATALOG, R1.SPECIFIC_SCHEMA, R1.SPECIFIC_NAME ) )
  WHERE ( MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME ) IS NULL
          AND
            ( P1.SPECIFIC_CATALOG, P1.SPECIFIC_SCHEMA, P1.SPECIFIC_NAME ) IN
            ( SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME
              FROM DEFINITION_SCHEMA.ROUTINE_PRIVILEGES
              WHERE GRANTEE IN
                      ( 'PUBLIC', CURRENT_USER )
                   OR
                      GRANTEE IN
                      ( SELECT ROLE_NAME
                        FROM ENABLED_ROLES ) )
    AND P1.SPECIFIC_CATALOG
      = ( SELECT CATALOG_NAME
          FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE PARAMETERS
  TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

No additional Conformance Rules.

20.2 ROUTINES view

This Subclause modifies Subclause 5.48, “ROUTINES view”, in ISO/IEC 9075-11.

Function

Identify the SQL-invoked routines in this catalog that are accessible to a given user or role.

Definition

Replace the ROUTINES view with the following

```
CREATE VIEW ROUTINES AS
    SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
           ROUTINE_CATALOG, ROUTINE_SCHEMA, ROUTINE_NAME, ROUTINE_TYPE,
           MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME,
           R.USER_DEFINED_TYPE_CATALOG AS UDT_CATALOG,
           R.USER_DEFINED_TYPE_SCHEMA AS UDT_SCHEMA,
           R.USER_DEFINED_TYPE_NAME AS UDT_NAME,
           D.DATA_TYPE, D.CHARACTER_MAXIMUM_LENGTH, D.CHARACTER_OCTET_LENGTH,
           D.CHARACTER_SET_CATALOG, D.CHARACTER_SET_SCHEMA, D.CHARACTER_SET_NAME,
           D.COLLATION_CATALOG, D.COLLATION_SCHEMA, D.COLLATION_NAME,
           D.NUMERIC_PRECISION, D.NUMERIC_PRECISION_RADIX, D.NUMERIC_SCALE,
           D.DATETIME_PRECISION, D.INTERVAL_TYPE, D.INTERVAL_PRECISION,
           D.USER_DEFINED_TYPE_CATALOG AS TYPE_UDT_CATALOG,
           D.USER_DEFINED_TYPE_SCHEMA AS TYPE_UDT_SCHEMA,
           D.USER_DEFINED_TYPE_NAME AS TYPE_UDT_NAME,
           D.SCOPE_CATALOG, D.SCOPE_SCHEMA, D.SCOPE_NAME,
           D.MAXIMUM_CARDINALITY, D.DTD_IDENTIFIER, ROUTINE_BODY,
           CASE
             WHEN EXISTS
                ( SELECT *
                  FROM DEFINITION_SCHEMA.SCHEMATA AS S
                  WHERE ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA )
                      = ( S.CATALOG_NAME, S.SCHEMA_NAME )
                    AND
                        ( SCHEMA_OWNER =
                          CURRENT_USER
                        OR
                          SCHEMA_OWNER IN
                          ( SELECT ROLE_NAME
                            FROM ENABLED_ROLES ) ) )
             THEN ROUTINE_DEFINITION
             ELSE NULL
           END AS ROUTINE_DEFINITION,
           EXTERNAL_NAME, EXTERNAL_LANGUAGE, PARAMETER_STYLE,
           IS_DETERMINISTIC, SQL_DATA_ACCESS, IS_NULL_CALL, SQL_PATH,
           SCHEMA_LEVEL_ROUTINE, MAX_DYNAMIC_RESULT_SETS,
           IS_USER_DEFINED_CAST, IS_IMPLICITLY_INVOCABLE, SECURITY_TYPE,
           TO_SQL_SPECIFIC_CATALOG, TO_SQL_SPECIFIC_SCHEMA, TO_SQL_SPECIFIC_NAME,
           AS_LOCATOR, CREATED, LAST_ALTERED, NEW_SAVEPOINT_LEVEL,
           IS_UDT_DEPENDENT, DT.DATA_TYPE AS RESULT_CAST_FROM_DATA_TYPE,
           RESULT_CAST_AS_LOCATOR, DT.CHARACTER_MAXIMUM_LENGTH AS
RESULT_CAST_CHAR_MAX_LENGTH,
           DT.CHARACTER_OCTET_LENGTH AS RESULT_CAST_CHAR_OCTET_LENGTH,
           DT.CHARACTER_SET_CATALOG AS RESULT_CAST_CHAR_SET_CATALOG,
           DT.CHARACTER_SET_SCHEMA AS RESULT_CAST_CHAR_SET_SCHEMA,
           DT.CHARACTER_SET_NAME AS RESULT_CAST_CHARACTER_SET_NAME,
           DT.COLLATION_CATALOG AS RESULT_CAST_COLLATION_CATALOG,
           DT.COLLATION_SCHEMA AS RESULT_CAST_COLLATION_SCHEMA,
           DT.COLLATION_NAME AS RESULT_CAST_COLLATION_NAME,
           DT.NUMERIC_PRECISION AS RESULT_CAST_NUMERIC_PRECISION,
           DT.NUMERIC_PRECISION_RADIX AS RESULT_CAST_NUMERIC_RADIX,
           DT.NUMERIC_SCALE AS RESULT_CAST_NUMERIC_SCALE,
           DT.DATETIME_PRECISION AS RESULT_CAST_DATETIME_PRECISION,
           DT.INTERVAL_TYPE AS RESULT_CAST_INTERVAL_TYPE,
           DT.INTERVAL_PRECISION AS RESULT_CAST_INTERVAL_PRECISION,
           DT.USER_DEFINED_TYPE_CATALOG AS RESULT_CAST_TYPE_UDT_CATALOG,
           DT.USER_DEFINED_TYPE_SCHEMA AS RESULT_CAST_TYPE_UDT_SCHEMA,
           DT.USER_DEFINED_TYPE_NAME AS RESULT_CAST_TYPE_UDT_NAME,
           DT.SCOPE_CATALOG AS RESULT_CAST_SCOPE_CATALOG,
           DT.SCOPE_SCHEMA AS RESULT_CAST_SCOPE_SCHEMA,
           DT.SCOPE_NAME AS RESULT_CAST_SCOPE_NAME,
           DT.MAXIMUM_CARDINALITY AS RESULT_CAST_MAX_CARDINALITY,
           DT.DTD_IDENTIFIER AS RESULT_CAST_DTD_IDENTIFIER,
           D2.DATA_TYPE AS XML_CHAR_TYPE,
           D2.CHARACTER_MAXIMUM_LENGTH AS XML_CHAR_MAX_LEN,
           D2.CHARACTER_OCTET_LENGTH AS XML_CHAR_OCT_LEN,
           D2.CHARACTER_SET_CATALOG AS XML_CHAR_SET_CAT,
           D2.CHARACTER_SET_SCHEMA AS XML_CHAR_SET_SCH,
           D2.CHARACTER_SET_NAME AS XML_CHAR_SET_NAME,
           D2.COLLATION_CATALOG AS XML_CHAR_COLL_CAT,
           D2.COLLATION_SCHEMA AS XML_CHAR_COLL_SCH,
           D2.COLLATION_NAME AS XML_CHAR_COLL_NAME,
           D2.DTD_IDENTIFIER AS XML_CHAR_DTD_ID,
           R.XML_OPTION
    FROM ( ( DEFINITION_SCHEMA.ROUTINES R
      LEFT JOIN
           DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR D
        ON ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
             'ROUTINE', R.DTD_IDENTIFIER )
         = ( D.OBJECT_CATALOG, D.OBJECT_SCHEMA, D.OBJECT_NAME,
             D.OBJECT_TYPE, D.DTD_IDENTIFIER )
      LEFT JOIN
           DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS DT
        ON ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
             'ROUTINE', RESULT_CAST_FROM_DTD_IDENTIFIER )
         = ( DT.OBJECT_CATALOG, DT.OBJECT_SCHEMA, DT.OBJECT_NAME,
             DT.OBJECT_TYPE, DT.DTD_IDENTIFIER ) )
      LEFT JOIN
           DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR D2
        ON ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
             'ROUTINE', R.XML_STRING_DTD_IDENTIFIER )
         = ( D2.OBJECT_CATALOG, D2.OBJECT_SCHEMA, D2.OBJECT_NAME,
             D2.OBJECT_TYPE, D2.DTD_IDENTIFIER ) )
    WHERE ( MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME) IS NULL
              AND
              ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME ) IN
```

```
                ( SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME
                  FROM DEFINITION_SCHEMA.ROUTINE_PRIVILEGES
                  WHERE GRANTEE IN
                          ( 'PUBLIC', CURRENT_USER )
                       OR
                          GRANTEE IN
                          ( SELECT ROLE_NAME
                            FROM ENABLED_ROLES ) )
      AND SPECIFIC_CATALOG
        = ( SELECT CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE ROUTINES
    TO PUBLIC WITH GRANT OPTION;
```

NOTE 28 — The ROUTINES view contains three sets of columns that each describe a data type. The set of columns that are prefixed with “XML_” decribes the associated string type, if any, of the result of the <SQL-invoked routine>, if its declared type is XML. The set of columns that are prefixed with “RESULT_CAST_” describes the data type specified in the <result cast>, if any, contained in the <SQL-invoked routine>. The third set of columns describes the data type specified in the <returns data type> contained in the <SQL-invoked routine>.

Conformance Rules

No additional Conformance Rules.

20.3 Short name views

This Subclause modifies Subclause 5.77, “Short name views”, in ISO/IEC 9075-11.

Function

Provide alternative views that use only identifiers that do not require Feature F391, “Long identifiers”.

Definition

Replace the PARAMETERS_S view with the following

```
CREATE VIEW PARAMETERS_S
         ( SPECIFIC_CATALOG,   SPECIFIC_SCHEMA,    SPECIFIC_NAME,
           ORDINAL_POSITION,   PARAMETER_MODE,     IS_RESULT,
           AS_LOCATOR,         PARAMETER_NAME,     FROM_SQL_SPEC_CAT,
           FROM_SQL_SPEC_SCH,  FROM_SQL_SPEC_NAME, TO_SQL_SPEC_CAT,
           TO_SQL_SPEC_SCHEMA, TO_SQL_SPEC_NAME,   DATA_TYPE,
           CHAR_MAX_LENGTH,    CHAR_OCTET_LENGTH,  CHAR_SET_CATALOG,
           CHAR_SET_SCHEMA,    CHARACTER_SET_NAME, COLLATION_CATALOG,
           COLLATION_SCHEMA,   COLLATION_NAME,     NUMERIC_PRECISION,
           NUMERIC_PREC_RADIX, NUMERIC_SCALE,      DATETIME_PRECISION,
           INTERVAL_TYPE,      INTERVAL_PRECISION, UDT_CATALOG,
           UDT_SCHEMA,         UDT_NAME,           SCOPE_CATALOG,
           SCOPE_SCHEMA,       SCOPE_NAME,         MAX_CARDINALITY,
           DTD_IDENTIFIER,     XML_CHAR_TYPE,      XML_CHAR_MAX_LEN,
           XML_CHAR_OCT_LEN,   XML_CHAR_SET_CAT,   XML_CHAR_SET_SCH,
           XML_CHAR_SET_NAME,  XML_CHAR_COLL_CAT,  XML_CHAR_COLL_SCH,
           XML_CHAR_COLL_NAME, XML_CHAR_DTD_ID,    XML_OPTION ) AS
    SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
           ORDINAL_POSITION, PARAMETER_MODE, IS_RESULT,
           AS_LOCATOR, PARAMETER_NAME, FROM_SQL_SPECIFIC_CATALOG,
           FROM_SQL_SPECIFIC_SCHEMA, FROM_SQL_SPECIFIC_NAME, TO_SQL_SPECIFIC_CATALOG,
           TO_SQL_SPECIFIC_SCHEMA, TO_SQL_SPECIFIC_NAME, DATA_TYPE,
           CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH, CHARACTER_SET_CATALOG,
           CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, COLLATION_CATALOG,
           COLLATION_SCHEMA, COLLATION_NAME, NUMERIC_PRECISION,
           NUMERIC_PRECISION_RADIX, NUMERIC_SCALE, DATETIME_PRECISION,
           INTERVAL_TYPE, INTERVAL_PRECISION, UDT_CATALOG,
           UDT_SCHEMA, UDT_NAME, SCOPE_CATALOG,
           SCOPE_SCHEMA, SCOPE_NAME, MAXIMUM_CARDINALITY,
           DTD_IDENTIFIER, XML_CHAR_TYPE, XML_CHAR_MAX_LEN,
           XML_CHAR_OCT_LEN, XML_CHAR_SET_CAT, XML_CHAR_SET_SCH,
           XML_CHAR_SET_NAME, XML_CHAR_COLL_CAT, XML_CHAR_COLL_SCH,
           XML_CHAR_COLL_NAME, XML_CHAR_DTD_ID, XML_OPTION
    FROM INFORMATION_SCHEMA.PARAMETERS;

GRANT SELECT ON TABLE PARAMETERS_S
    TO PUBLIC WITH GRANT OPTION;
```

Replace the ROUTINES_S view with the following

```
CREATE VIEW ROUTINES_S
         ( SPECIFIC_CATALOG,   SPECIFIC_SCHEMA,    SPECIFIC_NAME,
           ROUTINE_CATALOG,    ROUTINE_SCHEMA,     ROUTINE_NAME,
           ROUTINE_TYPE,       MODULE_CATALOG,     MODULE_SCHEMA,
           MODULE_NAME,        UDT_CATALOG,        UDT_SCHEMA,
           UDT_NAME,           DATA_TYPE,          CHAR_MAX_LENGTH,
           CHAR_OCTET_LENGTH,  CHAR_SET_CATALOG,   CHAR_SET_SCHEMA,
           CHARACTER_SET_NAME, COLLATION_CATALOG,  COLLATION_SCHEMA,
           COLLATION_NAME,     NUMERIC_PRECISION,  NUMERIC_PREC_RADIX,
           NUMERIC_SCALE,      DATETIME_PRECISION, INTERVAL_TYPE,
           INTERVAL_PRECISION, TYPE_UDT_CATALOG,   TYPE_UDT_SCHEMA,
           TYPE_UDT_NAME,      SCOPE_CATALOG,      SCOPE_SCHEMA,
           SCOPE_NAME,         MAX_CARDINALITY,    DTD_IDENTIFIER,
           ROUTINE_BODY,       ROUTINE_DEFINITION, EXTERNAL_NAME,
           EXTERNAL_LANGUAGE,  PARAMETER_STYLE,    IS_DETERMINISTIC,
           SQL_DATA_ACCESS,    IS_NULL_CALL,       SQL_PATH,
           SCH_LEVEL_ROUTINE,  MAX_DYN_RESLT_SETS, IS_USER_DEFND_CAST,
           IS_IMP_INVOCABLE,   SECURITY_TYPE,      TO_SQL_SPEC_CAT,
           TO_SQL_SPEC_SCHEMA, TO_SQL_SPEC_NAME,   AS_LOCATOR,
           CREATED,            LAST_ALTERED,       NEW_SAVEPOINT_LVL,
           IS_UDT_DEPENDENT,   RC_FROM_DATA_TYPE,  RC_AS_LOCATOR,
           RC_CHAR_MAX_LENGTH, RC_CHAR_OCT_LENGTH, RC_CHAR_SET_CAT,
           RC_CHAR_SET_SCHEMA, RC_CHAR_SET_NAME,   RC_COLLATION_CAT,
           RC_COLLATION_SCH,   RC_COLLATION_NAME,  RC_NUMERIC_PREC,
           RC_NUMERIC_RADIX,   RC_NUMERIC_SCALE,   RC_DATETIME_PREC,
           RC_INTERVAL_TYPE,   RC_INTERVAL_PREC,   RC_TYPE_UDT_CAT,
           RC_TYPE_UDT_SCHEMA, RC_TYPE_UDT_NAME,   RC_SCOPE_CATALOG,
           RC_SCOPE_SCHEMA,    RC_SCOPE_NAME,      RC_MAX_CARDINALITY,
           RC_DTD_IDENTIFIER,  XML_CHAR_TYPE,      XML_CHAR_MAX_LEN,
           XML_CHAR_OCT_LEN,   XML_CHAR_SET_CAT,   XML_CHAR_SET_SCH,
           XML_CHAR_SET_NAME,  XML_CHAR_COLL_CAT,  XML_CHAR_COLL_SCH,
           XML_CHAR_COLL_NAME, XML_CHAR_DTD_ID,    XML_OPTION ) AS
    SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
           ROUTINE_CATALOG, ROUTINE_SCHEMA, ROUTINE_NAME,
           ROUTINE_TYPE, MODULE_CATALOG, MODULE_SCHEMA,
           MODULE_NAME, UDT_CATALOG, UDT_SCHEMA,
           UDT_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH,
           CHARACTER_OCTET_LENGTH, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA,
           CHARACTER_SET_NAME, COLLATION_CATALOG, COLLATION_SCHEMA,
           COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
           NUMERIC_SCALE, DATETIME_PRECISION, INTERVAL_TYPE,
           INTERVAL_PRECISION, TYPE_UDT_CATALOG, TYPE_UDT_SCHEMA,
           TYPE_UDT_NAME, SCOPE_CATALOG, SCOPE_SCHEMA,
           SCOPE_NAME, MAXIMUM_CARDINALITY, DTD_IDENTIFIER,
           ROUTINE_BODY, ROUTINE_DEFINITION, EXTERNAL_NAME,
           EXTERNAL_LANGUAGE, PARAMETER_STYLE, IS_DETERMINISTIC,
           SQL_DATA_ACCESS, IS_NULL_CALL, SQL_PATH,
           SCHEMA_LEVEL_ROUTINE, MAX_DYNAMIC_RESULT_SETS, IS_USER_DEFINED_CAST,
           IS_IMPLICITLY_INVOCABLE, SECURITY_TYPE, TO_SQL_SPECIFIC_CATALOG,
           TO_SQL_SPECIFIC_SCHEMA, TO_SQL_SPECIFIC_NAME, AS_LOCATOR,
           CREATED, LAST_ALTERED, NEW_SAVEPOINT_LEVEL,
           IS_UDT_DEPENDENT, RESULT_CAST_FROM_DATA_TYPE, RESULT_CAST_AS_LOCATOR,
           RESULT_CAST_CHAR_MAX_LENGTH, RESULT_CAST_CHAR_OCTET_LENGTH,
RESULT_CAST_CHAR_SET_CATALOG,
           RESULT_CAST_CHAR_SET_SCHEMA, RESULT_CAST_CHAR_SET_NAME,
RESULT_CAST_COLLATION_CATALOG,
           RESULT_CAST_COLLATION_SCHEMA, RESULT_CAST_COLLATION_NAME,
RESULT_CAST_NUMERIC_PRECISION,
           RESULT_CAST_NUMERIC_RADIX, RESULT_CAST_NUMERIC_SCALE,
RESULT_CAST_DATETIME_PRECISION,
           RESULT_CAST_INTERVAL_TYPE, RESULT_CAST_INTERVAL_PRECISION,
RESULT_CAST_TYPE_UDT_CATALOG,
          RESULT_CAST_TYPE_UDT_SCHEMA, RESULT_CAST_TYPE_UDT_NAME, RESULT_CAST_SCOPE_CATALOG,

           RESULT_CAST_SCOPE_SCHEMA, RESULT_CAST_SCOPE_NAME, RESULT_CAST_MAX_CARDINALITY,
           RESULT_CAST_DTD_IDENTIFIER, XML_CHAR_TYPE, XML_CHAR_MAX_LEN,
           XML_CHAR_OCT_LEN, XML_CHAR_SET_CAT, XML_CHAR_SET_SCH,
           XML_CHAR_SET_NAME, XML_CHAR_COLL_CAT, XML_CHAR_COLL_SCH,
           XML_CHAR_COLL_NAME, XML_CHAR_DTD_ID, XML_OPTION
    FROM INFORMATION_SCHEMA.ROUTINES;

GRANT SELECT ON TABLE ROUTINES_S
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

No additional Conformance Rules.

21 Definition Schema

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

21.1 DATA_TYPE_DESCRIPTOR base table

This Subclause modifies Subclause 6.21, "DATA_TYPE_DESCRIPTOR base table", in ISO/IEC 9075-11.

Function

The DATA_TYPE_DESCRIPTOR table has one row for each domain, one row for each column (in each table) and for each attribute (in each structured type) that is defined as having a data type rather than a domain, one row for each distinct type, one row for the result type of each SQL-invoked function, one row for each SQL parameter of each SQL-invoked routine, one row for the result type of each method specification, one row for each parameter of each method specification, one row for each structured type whose associated reference type has a user-defined representation, one row for each field in a row type, one row for each element type of a collection type, one row for each referenced type of a reference type, and one row for each maximal supertype of each field, element type, or referenced type. It effectively contains a representation of the data type descriptors.

Definition

Augment constraint DATA_TYPE_DESCRIPTOR_DATA_TYPE_CHECK_COMBINATIONS:

Add the following OR clause to the end of the constraint:

```
OR
  ( DATA_TYPE = 'XML'
  AND
    ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
      NUMERIC_SCALE, DATETIME_PRECISION, CHARACTER_OCTET_LENGTH,
      CHARACTER_MAXIMUM_LENGTH, INTERVAL_TYPE, INTERVAL_PRECISION )
        IS NULL
  AND
    ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
      USER_DEFINED_TYPE_NAME ) IS NULL
  AND
    ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
  AND
    MAXIMUM_CARDINALITY IS NULL )
```

Alter the final OR clause of the constraint:

```
OR
  ( DATA_TYPE NOT IN
                ( 'CHARACTER', 'CHARACTER VARYING', 'CHARACTER LARGE OBJECT',
                  'BINARY LARGE OBJECT',
                  'NUMERIC', 'DECIMAL', 'SMALLINT', 'INTEGER', 'BIGINT',
                  'FLOAT', 'REAL', 'DOUBLE PRECISION',
                  'DATE', 'TIME', 'TIMESTAMP',
                  'INTERVAL', 'BOOLEAN', 'USER-DEFINED',
                  'REF', 'ROW', 'ARRAY', 'MULTISET', 'XML' ) ) ),
```

Description

1) $\boxed{\text{Insert this Description}}$ If DATA_TYPE is 'XML', then the data type being described is the XML type.

21.2 PARAMETERS base table

This Subclause modifies Subclause 6.31, “PARAMETERS base table”, in ISO/IEC 9075-11.

Function

The PARAMETERS table has one row for each SQL parameter of each SQL-invoked routine described in the ROUTINES base table.

Definition

Add the following column definitions

```
XML_STRING_DTD_IDENTIFIER  INFORMATION_SCHEMA.SQL_IDENTIFIER,
XML_OPTION                 INFORMATION_SCHEMA.CHARACTER_DATA
  CONSTRAINT XML_OPTION_CHECK
    CHECK (XML_OPTION IN ( 'CONTENT', 'DOCUMENT' ) ),
```

Description

1) Insert this Description SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME, and XML_STRING_DTD_IDENTIFIER are the values of OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, and DTD_IDENTIFIER, respectively, of the row in DATA_TYPE_DESCRIPTOR that describes the associated string type of the parameter being described.

2) Insert this Description The value of XML_OPTION is the associated XML option of the parameter being described.

21.3 ROUTINES base table

This Subclause modifies Subclause 6.40, “ROUTINES base table”, in ISO/IEC 9075-11.

Function

The ROUTINES base table has one row for each SQL-invoked routine.

Definition

Add the following column definitions

```
XML_STRING_DTD_IDENTIFIER  INFORMATION_SCHEMA.SQL_IDENTIFIER,
XML_OPTION                 INFORMATION_SCHEMA.CHARACTER_DATA
  CONSTRAINT XML_OPTION_CHECK
    CHECK (XML_OPTION IN ( 'CONTENT', 'DOCUMENT' ) ),
```

Description

1) Insert this Description SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME, and XML_STRING_DTD_IDENTIFIER are the values of OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, and DTD_IDENTIFIER, respectively, of the row in DATA_TYPE_DESCRIPTOR that describes the associated string type of the result of SQL-invoked routine being described.

2) Insert this Description The value of XML_OPTION is the associated XML option of the result of SQL-invoked routine being described.

22 The SQL/XML XML Schema

22.1 The SQL/XML XML Schema

Function

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

Syntax Rules

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

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

```
        <xsd:enumeration value="DOUBLE PRECISION"/>
        <xsd:enumeration value="BOOLEAN"/>
        <xsd:enumeration value="DATE"/>
        <xsd:enumeration value="TIME"/>
        <xsd:enumeration value="TIME WITH TIME ZONE"/>
        <xsd:enumeration value="TIMESTAMP"/>
        <xsd:enumeration value="TIMESTAMP WITH TIME ZONE"/>
        <xsd:enumeration value="INTERVAL YEAR"/>
        <xsd:enumeration value="INTERVAL YEAR TO MONTH"/>
        <xsd:enumeration value="INTERVAL MONTH"/>
        <xsd:enumeration value="INTERVAL DAY"/>
        <xsd:enumeration value="INTERVAL DAY TO HOUR"/>
        <xsd:enumeration value="INTERVAL DAY TO MINUTE"/>
        <xsd:enumeration value="INTERVAL DAY TO SECOND"/>
        <xsd:enumeration value="INTERVAL HOUR"/>
        <xsd:enumeration value="INTERVAL HOUR TO MINUTE"/>
        <xsd:enumeration value="INTERVAL HOUR TO SECOND"/>
        <xsd:enumeration value="INTERVAL MINUTE"/>
        <xsd:enumeration value="INTERVAL MINUTE TO SECOND"/>
        <xsd:enumeration value="INTERVAL SECOND"/>
        <xsd:enumeration value="XML"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="fieldType">
      <xsd:attribute name="name" type="xsd:string"/>
      <xsd:attribute name="mappedType" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="sqltype">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="field" type="sqlxml:fieldType"
                        minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="kind"
                        type="sqlxml:kindKeyword"/>
        <xsd:attribute name="name"
                        type="sqlxml:typeKeyword" use="optional"/>
        <xsd:attribute name="length" type="xsd:integer"
                        use="optional"/>
        <xsd:attribute name="maxLength" type="xsd:integer"
                        use="optional"/>
        <xsd:attribute name="characterSetName" type="xsd:string"
                        use="optional"/>
        <xsd:attribute name="collation" type="xsd:string"
                        use="optional"/>
        <xsd:attribute name="precision" type="xsd:integer"
                        use="optional"/>
        <xsd:attribute name="scale" type="xsd:integer"
                        use="optional"/>
        <xsd:attribute name="maxExponent" type="xsd:integer"
                        use="optional"/>
        <xsd:attribute name="minExponent" type="xsd:integer"
                        use="optional"/>
        <xsd:attribute name="userPrecision" type="xsd:integer"
                        use="optional"/>
        <xsd:attribute name="leadingPrecision" type="xsd:integer"
```

```
                        use="optional"/>
      <xsd:attribute name="maxElements" type="xsd:integer"
                        use="optional"/>
      <xsd:attribute name="catalogName" type="xsd:string"
                        use="optional"/>
      <xsd:attribute name="schemaName" type="xsd:string"
                        use="optional"/>
      <xsd:attribute name="domainName" type="xsd:string"
                        use="optional"/>
      <xsd:attribute name="typeName" type="xsd:string"
                        use="optional"/>
      <xsd:attribute name="mappedType" type="xsd:string"
                        use="optional"/>
      <xsd:attribute name="mappedElementType" type="xsd:string"
                        use="optional"/>
      <xsd:attribute name="final" type="xsd:boolean"
                        use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="objectType">
     <xsd:restriction base="xsd:string">
        <xsd:enumeration value="CATALOG" />
        <xsd:enumeration value="SCHEMA" />
        <xsd:enumeration value="BASE TABLE" />
        <xsd:enumeration value="VIEWED TABLE" />
        <xsd:enumeration value="CHARACTER SET" />
        <xsd:enumeration value="COLLATION" />
     </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="sqlname">
     <xsd:complexType>
        <xsd:attribute name="type" type="sqlxml:objectType"
                          use="required" />
        <xsd:attribute name="catalogName" type="xsd:string" />
        <xsd:attribute name="schemaName" type="xsd:string" />
        <xsd:attribute name="localName" type="xsd:string" />
     </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

General Rules

None.

Conformance Rules

None.

This page intentionally left blank.

23 Status codes

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

23.1 SQLSTATE

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

Table 12, “SQLSTATE class and subclass values”, modifies Table 32, “SQLSTATE class and subclass values”, in ISO/IEC 9075-2.

Table 12 — SQLSTATE class and subclass values

Category	**Condition**	**Class**	**Subcondition**	**Subclass**
X	*data exception*	22	(*no subclass*)	000
			invalid comment	00S
			invalid processing instruction	00T
			invalid XML content	00N
			invalid XML document	00M
			nonidentical notations with the same name	00J
			nonidentical unparsed entities with the same name	00K
			not an XML document	00L
			XML value overflow	00R
X	*SQL/XML mapping error*	0N	(*no subclass*)	000
			unmappable XML Name	001
			invalid XML character	002
W	*warning*	01	(*no subclass*)	000
			column cannot be mapped	010

This page intentionally left blank.

24 Conformance

24.1 Claims of conformance to SQL/XML

In addition to the requirements of ISO/IEC 9075-1, Clause 8, “Conformance”, a claim of conformance to this part of ISO/IEC 9075 shall:

1) Claim conformance to Feature X010, “XML type”.
2) Claim conformance to Feature X031, “XMLElement”.
3) Claim conformance to Feature X032, “XMLForest”.
4) Claim conformance to Feature X034, “XMLAgg”.
5) Claim conformance to at least one of the following:
   a) Feature X070, “XMLSerialize: CONTENT option”.
   b) Feature X071, “XMLSerialize: DOCUMENT option”.
   c) Feature X100, “Host language support for XML: CONTENT option”, and Feature X110, “Host language support for XML: VARCHAR option”.
   d) Feature X100, “Host language support for XML: CONTENT option”, and Feature X111, “Host language support for XML: CLOB option”.
   e) Feature X101, “Host language support for XML: DOCUMENT option”, and Feature X110, “Host language support for XML: VARCHAR option”.
   f) Feature X101, “Host language support for XML: DOCUMENT option”, and Feature X111, “Host language support for XML: CLOB option”.

24.2 Additional conformance requirements for SQL/XML

There are no additional conformance requirements for this part of ISO/IEC 9075.

24.3 Implied feature relationships of SQL/XML

Table 13 — Implied feature relationships of SQL/XML

Feature ID	Feature Name	Implied Feature ID	Implied Feature Name
X011	Arrays of XML type	X010	XML type
X011	Arrays of XML type	S091	Basic array support
X012	Multisets of XML type	X010	XML type
X012	Multisets of XML type	S271	Basic multiset support
X013	Distinct types of XML	X010	XML type
X014	Attributes of XML type	X010	XML type
X014	Attributes of XML type	S023	Basic structured types
X015	Fields of XML type	X010	XML type
X015	Fields of XML type	T051	Row types
X016	Persistent XML values	X010	XML type
X020	XML concatenation	X010	XML type
X031	XMLElement	X010	XML type
X032	XMLForest	X010	XML type
X033	XMLRoot	X010	XML type
X034	XMLAgg	X010	XML type
X035	XMLAgg: ORDER BY option	X034	XMLAgg
X036	XMLCOMMENT	X010	XML type
X037	XMLPI	X010	XML type
X048	Basic table mapping: base64 encoding of binary strings	X010	XML type
X049	Basic table mapping: hex encoding of binary strings	X010	XML type

Feature ID	Feature Name	Implied Feature ID	Implied Feature Name
X051	Advanced table mapping: null absent	X041	Basic table mapping: null absent
X052	Advanced table mapping: null as nil	X042	Basic table mapping: null as nil
X053	Advanced table mapping: table as forest	X043	Basic table mapping: table as forest
X054	Advanced table mapping: table as element	X044	Basic table mapping: table as element
X055	Advanced table mapping: with target namespace	X045	Basic table mapping: with target namespace
X056	Advanced table mapping: data mapping	X046	Basic table mapping: data mapping
X057	Advanced table mapping: metadata mapping	X047	Basic table mapping: metadata mapping
X058	Advanced table mapping: base64 encoding of binary strings	X010	XML type
X059	Advanced table mapping: hex encoding of binary strings	X010	XML type
X060	XMLParse: CONTENT option	X010	XML type
X061	XMLParse: DOCUMENT option	X010	XML type
X070	XMLSerialize: CONTENT option	X010	XML type
X071	XMLSerialize: DOCUMENT option	X010	XML type
X080	Namespaces in XML publishing	X010	XML type
X081	Query-level XML namespace declarations	X010	XML type
X082	XML namespace declarations in DML	X010	XML type
X083	XML namespace declarations in DDL	X010	XML type
X084	XML namespace declarations in compound statements	X010	XML type
X090	XML document predicate	X010	XML type

Feature ID	Feature Name	Implied Feature ID	Implied Feature Name
X100	Host language support for XML: CONTENT option	X010	XML type
X101	Host language support for XML: DOCUMENT option	X010	XML type
X110	Host language support for XML: VARCHAR mapping	X010	XML type
X111	Host language support for XML: CLOB mapping	X010	XML type
X111	Host language support for XML: CLOB mapping	T041	Basic LOB data type support
X120	XML parameters in SQL routines	X010	XML type
X121	XML parameters in external routines	X010	XML type
X131	Query-level XMLBINARY clause	X010	XML type
X132	XMLBINARY clause in DML	X010	XML type
X133	XMLBINARY clause in DDL	X010	XML type
X134	XMLBINARY clause in compound statements	X010	XML type
X135	XMLBINARY clause in subqueries	X131	Query-level XMLBINARY clause

Annex A

(informative)

SQL Conformance Summary

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

The contents of this Annex summarizes all Conformance Rules, ordered by Feature ID and by Subclause.

1) Specifications for Feature F761, "Session management":
   a) Subclause 16.1, "<set XML option statement>":
      i) Without Feature F761, "Session management", conforming SQL language shall not contain a <set XML option statement>.
2) Specifications for Feature X010, "XML type":
   a) Subclause 6.1, "<data type>":
      i) Insert this CR Without Feature X010, "XML type", conforming SQL language shall not contain an <XML type>.
   b) Subclause 6.3, "<cast specification>":
      i) Insert this CR Without Feature X010, "XML type", conforming SQL language shall not contain a <cast operand> whose declared type is XML.
   c) Subclause 6.6, "<XML value expression>":
      i) Without Feature X010, "XML type", conforming SQL language shall not contain an <XML value expression>.
   d) Subclause 6.7, "<XML value function>":
      i) Without Feature X010, "XML type", conforming SQL language shall not contain an <XML value function>.
3) Specifications for Feature X011, "Arrays of XML type":
   a) Subclause 6.1, "<data type>":
      i) Insert this CR Without Feature X011, "Arrays of XML type", conforming SQL language shall not contain an <array type> that is based on a <data type> that is either the XML type or a distinct type whose source type is the XML type.
4) Specifications for Feature X012, "Multisets of XML type":
   a) Subclause 6.1, "<data type>":

i) Insert this CR Without Feature X012, “Multisets of XML type”, conforming SQL language shall not contain a <multiset type> that is based on a <data type> that is either the XML type or a distinct type whose source type is the XML type.

5) Specifications for Feature X013, “Distinct types on XML type”:

a) Subclause 12.5, “<user-defined type definition>”:

i) Insert this CR Without Feature X013, “Distinct types on XML type”, conforming SQL language shall not contain a <representation> that is a <predefined type> that is XML.

6) Specifications for Feature X014, “Attributes of XML type”:

a) Subclause 12.6, “<attribute definition>”:

i) Insert this CR Without Feature X014, “Attributes of XML type”, conforming SQL language shall not contain an <attribute definition> that contains a <data type> that is based on either the XML type or a distinct type whose source type is the XML type.

7) Specifications for Feature X015, “Fields of XML type”:

a) Subclause 6.2, “<field definition>”:

i) Insert this CR Without Feature X015, “Fields of XML type”, conforming SQL language shall not contain a <field definition> that contains a <data type> that is based on either the XML type or a distinct type whose source type is the XML type.

8) Specifications for Feature X016, “Persistent XML values”:

a) Subclause 12.1, “<column definition>”:

i) Insert this CR Without Feature X016, “Persistent XML values”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML type or a distinct type whose source type is the XML type.

b) Subclause 12.3, “<view definition>”:

i) Insert this CR Without Feature X016, “Persistent XML values”, conforming SQL language shall not contain a <view definition> that defines a column whose declared type is based on either the XML type or a distinct type whose source type is the XML type.

9) Specifications for Feature X020, “XML concatenation”:

a) Subclause 6.9, “<XML concatenation>”:

i) Without Feature X020, “XML concatenation”, conforming SQL language shall not contain an <XML concatenation>.

10) Specifications for Feature X031, “XMLElement”:

a) Subclause 6.10, “<XML element>”:

i) Without Feature X031, “XMLElement”, conforming SQL language shall not contain an <XML element>.

11) Specifications for Feature X032, “XMLForest”:

a) Subclause 6.11, “<XML forest>”:

i) Without Feature X032, “XMLForest”, conforming SQL language shall not contain an <XML forest>.

12) Specifications for Feature X033, “XMLRoot”:

a) Subclause 6.14, “<XML root>”:

i) Without Feature X033, “XMLRoot”, conforming SQL language shall not contain <XML root>.

13) Specifications for Feature X034, “XMLAgg”:

a) Subclause 11.2, “<aggregate function>”:

i) Insert this CR Without Feature X034, “XMLAgg”, conforming SQL language shall not contain an <XML aggregate>.

14) Specifications for Feature X035, “XMLAgg: ORDER BY option”:

a) Subclause 11.2, “<aggregate function>”:

i) Insert this CR Without Feature X035, “XMLAgg: ORDER BY option”, conforming SQL language shall not contain an <XML aggregate> that contains a <sort specification list>.

15) Specifications for Feature X036, “XMLComment”:

a) Subclause 6.8, “<XML comment>”:

i) Without Feature X036, “XMLComment”, conforming SQL language shall not contain an <XML comment>.

16) Specifications for Feature X037, “XMLPI”:

a) Subclause 6.12, “<XML PI>”:

i) Without Feature X037, “XMLPI”, conforming SQL language shall not contain an <XML PI>.

17) Specifications for Feature X041, “Basic table mapping: nulls absent”:

a) Subclause 9.3, “Mapping an SQL table to XML and an XML Schema document”:

i) Without Feature X041, “Basic table mapping: nulls absent”, a conforming application shall not invoke this Subclause of this part of this International Standard with *NULLS* set to indicate that nulls are mapped to elements that are marked to absent elements.

18) Specifications for Feature X042, “Basic table mapping: null as nil”:

a) Subclause 9.3, “Mapping an SQL table to XML and an XML Schema document”:

i) Without Feature X042, “Basic table mapping: null as nil”, a conforming application shall not invoke this Subclause of this part of this International Standard with *NULLS* set to indicate that nulls are mapped to elements that are marked with `xsi:nil="true"`.

19) Specifications for Feature X043, “Basic table mapping: table as forest”:

a) Subclause 9.3, “Mapping an SQL table to XML and an XML Schema document”:

i) Without Feature X043, “Basic table mapping: table as forest”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TABLEFOREST* set to *True*.

20) Specifications for Feature X044, “Basic table mapping: table as element”:

a) Subclause 9.3, “Mapping an SQL table to XML and an XML Schema document”:

i) Without Feature X044, “Basic table mapping: table as element”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TABLEFOREST* set to *False*.

21) Specifications for Feature X045, “Basic table mapping: with target namespace”:

a) Subclause 9.3, “Mapping an SQL table to XML and an XML Schema document”:

i) Without Feature X045, “Basic table mapping: with target namespace”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TARGETNS* that is not a zero-length string.

22) Specifications for Feature X046, “Basic table mapping: data mapping”:

a) Subclause 9.3, “Mapping an SQL table to XML and an XML Schema document”:

i) Without Feature X046, “Basic table mapping: data mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with with *DATA* set to *True*.

23) Specifications for Feature X047, “Basic table mapping: metadata mapping”:

a) Subclause 9.3, “Mapping an SQL table to XML and an XML Schema document”:

i) Without Feature X047, “Basic table mapping: metadata mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with *METADATA* set to *True*.

24) Specifications for Feature X048, “Basic table mapping: base64 encoding of binary strings”:

a) Subclause 9.3, “Mapping an SQL table to XML and an XML Schema document”:

i) Without Feature X048, “Basic table mapping: base64 encoding of binary strings”, a conforming application shall not invoke this Subclause of this part of this International Standard with *ENCODING* set to indicate that binary strings are to be encoded using base64.

25) Specifications for Feature X049, “Basic table mapping: hex encoding of binary strings”:

a) Subclause 9.3, “Mapping an SQL table to XML and an XML Schema document”:

i) Without Feature X049, “Basic table mapping: hex encoding of binary strings”, a conforming application shall not invoke this Subclause of this part of this International Standard with *ENCODING* set to indicate that binary strings are to be encoded using hex.

26) Specifications for Feature X051, “Advanced table mapping: nulls absent”:

a) Subclause 9.4, “Mapping an SQL schema to an XML document and an XML Schema document”:

i) Without Feature X051, “Advanced table mapping: nulls absent”, a conforming application shall not invoke this Subclause of this part of this International Standard with *NULLS* set to indicate that nulls are mapped to elements that are marked to absent elements.

b) Subclause 9.5, “Mapping an SQL catalog to an XML document and an XML Schema document”:

i) Without Feature X051, “Advanced table mapping: nulls absent”, a conforming application shall not invoke this Subclause of this part of this International Standard with *NULLS* set to indicate that nulls are mapped to elements that are marked to absent elements.

27) Specifications for Feature X052, “Advanced table mapping: null as nil”:

a) Subclause 9.4, “Mapping an SQL schema to an XML document and an XML Schema document”:

i) Without Feature X052, “Advanced table mapping: null as nil”, a conforming application shall not invoke this Subclause of this part of this International Standard with *NULLS* set to indicate that nulls are mapped to elements that are marked with **`xsi:nil="true"`**.

b) Subclause 9.5, “Mapping an SQL catalog to an XML document and an XML Schema document”:

i) Without Feature X052, “Advanced table mapping: null as nil”, a conforming application shall not invoke this Subclause of this part of this International Standard with *NULLS* set to indicate that nulls are mapped to elements that are marked with **`xsi:nil="true"`**.

28) Specifications for Feature X053, “Advanced table mapping: table as forest”:

a) Subclause 9.4, “Mapping an SQL schema to an XML document and an XML Schema document”:

i) Without Feature X053, “Advanced table mapping: table as forest”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TABLEFOREST* set to *True*.

b) Subclause 9.5, “Mapping an SQL catalog to an XML document and an XML Schema document”:

i) Without Feature X053, “Advanced table mapping: table as forest”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TABLEFOREST* set to *True*.

29) Specifications for Feature X054, “Advanced table mapping: table as element”:

a) Subclause 9.4, “Mapping an SQL schema to an XML document and an XML Schema document”:

i) Without Feature X054, “Advanced table mapping: table as element”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TABLEFOREST* set to *False*.

b) Subclause 9.5, “Mapping an SQL catalog to an XML document and an XML Schema document”:

i) Without Feature X054, “Advanced table mapping: table as element”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TABLEFOREST* set to *False*.

30) Specifications for Feature X055, “Advanced table mapping: with target namespace”:

a) Subclause 9.4, “Mapping an SQL schema to an XML document and an XML Schema document”:

i) Without Feature X055, “Advanced table mapping: with target namespace”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TARGETNS* that is not a zero-length string.

b) Subclause 9.5, “Mapping an SQL catalog to an XML document and an XML Schema document”:

i) Without Feature X055, “Advanced table mapping: with target namespace”, a conforming application shall not invoke this Subclause of this part of this International Standard with *TARGETNS* that is not a zero-length string.

31) Specifications for Feature X056, “Advanced table mapping: data mapping”:

a) Subclause 9.4, “Mapping an SQL schema to an XML document and an XML Schema document”:

i) Without Feature X056, “Advanced table mapping: data mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with with *DATA* set to *True*.

b) Subclause 9.5, “Mapping an SQL catalog to an XML document and an XML Schema document”:

i) Without Feature X056, “Advanced table mapping: data mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with with *DATA* set to *True*.

32) Specifications for Feature X057, “Advanced table mapping: metadata mapping”:

a) Subclause 9.4, “Mapping an SQL schema to an XML document and an XML Schema document”:

i) Without Feature X057, “Advanced table mapping: metadata mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with *METADATA* set to *True*.

b) Subclause 9.5, “Mapping an SQL catalog to an XML document and an XML Schema document”:

i) Without Feature X057, “Advanced table mapping: metadata mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with *METADATA* set to *True*.

33) Specifications for Feature X058, “Advanced table mapping: base64 encoding of binary strings”:

a) Subclause 9.4, “Mapping an SQL schema to an XML document and an XML Schema document”:

i) Without Feature X058, “Advanced table mapping: base64 encoding of binary strings”, a conforming application shall not invoke this Subclause of this part of this International Standard with *ENCODING* set to indicate that binary strings are to be encoded using base64.

b) Subclause 9.5, “Mapping an SQL catalog to an XML document and an XML Schema document”:

i) Without Feature X058, “Advanced table mapping: base64 encoding of binary strings”, a conforming application shall not invoke this Subclause of this part of this International Standard with *ENCODING* set to indicate that binary strings are to be encoded using base64.

34) Specifications for Feature X059, “Advanced table mapping: hex encoding of binary strings”:

a) Subclause 9.4, “Mapping an SQL schema to an XML document and an XML Schema document”:

i) Without Feature X059, “Advanced table mapping: hex encoding of binary strings”, a conforming application shall not invoke this Subclause of this part of this International Standard with *ENCODING* set to indicate that binary strings are to be encoded using hex.

b) Subclause 9.5, “Mapping an SQL catalog to an XML document and an XML Schema document”:

i) Without Feature X059, “Advanced table mapping: hex encoding of binary strings”, a conforming application shall not invoke this Subclause of this part of this International Standard with *ENCODING* set to indicate that binary strings are to be encoded using hex.

35) Specifications for Feature X060, “XMLParse: CONTENT option”:

a) Subclause 6.13, “<XML parse>”:

i) Without Feature X060, “XMLParse: CONTENT option”, in conforming SQL language, <XML parse> shall not immediately contain a <document or content> that is CONTENT.

36) Specifications for Feature X061, “XMLParse: DOCUMENT option”:

a) Subclause 6.13, “<XML parse>”:

i) Without Feature X061, “XMLParse: DOCUMENT option”, in conforming SQL language, <XML parse> shall not immediately contain a <document or content> that is DOCUMENT.

37) Specifications for Feature X062, “XMLParse: explicit WHITESPACE option”:

a) Subclause 6.13, “<XML parse>”:

i) Without Feature X062, “XMLParse: explicit WHITESPACE option”, in conforming SQL language, <XML parse> shall not contain <XML whitespace option>.

38) Specifications for Feature X070, “XMLSerialize: CONTENT option”:

a) Subclause 6.5, “<string value function>”:

i) Without Feature X070, “XMLSerialize: CONTENT option”, in conforming SQL language, <XML serialize> shall not immediately contain a <document or content> that is CONTENT.

39) Specifications for Feature X071, “XMLSerialize: DOCUMENT option”:

a) Subclause 6.5, “<string value function>”:

i) Without Feature X071, “XMLSerialize: DOCUMENT option”, in conforming SQL language, <XML serialize> shall not immediately contain a <document or content> that is DOCUMENT.

40) Specifications for Feature X080, “Namespaces in XML publishing”:

a) Subclause 6.10, “<XML element>”:

i) Without Feature X080, “Namespaces in XML publishing”, in conforming SQL language, <XML element> shall not immediately contain <XML namespace declaration>.

b) Subclause 6.11, “<XML forest>”:

i) Without Feature X080, “Namespaces in XML publishing”, in conforming SQL language, <XML forest> shall not immediately contain <XML namespace declaration>.

41) Specifications for Feature X081, “Query-level XML namespace declarations”:

  a) Subclause 7.1, “<query expression>”:

    i) $\boxed{\text{Insert this CR}}$ Without Feature X081, “Query-level XML namespace declarations”, in conforming SQL language, <with clause> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

42) Specifications for Feature X082, “XML namespace declarations in DML”:

  a) Subclause 14.1, “<delete statement: searched>”:

    i) $\boxed{\text{Insert this CR}}$ Without Feature X082, “XML namespace declarations in DML”, in conforming SQL language, a <delete statement: searched> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

  b) Subclause 14.2, “<insert statement>”:

    i) $\boxed{\text{Insert this CR}}$ Without Feature X082, “XML namespace declarations in DML”, in conforming SQL language, an <insert statement> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

  c) Subclause 14.3, “<merge statement>”:

    i) $\boxed{\text{Insert this CR}}$ Without Feature X082, “XML namespace declarations in DML”, in conforming SQL language, a <merge statement> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

  d) Subclause 14.4, “<update statement: positioned>”:

    i) $\boxed{\text{Insert this CR}}$ Without Feature X082, “XML namespace declarations in DML”, in conforming SQL language, an <update statement: positioned> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

  e) Subclause 14.5, “<update statement: searched>”:

    i) $\boxed{\text{Insert this CR}}$ Without Feature X082, “XML namespace declarations in DML”, in conforming SQL language, an <update statement: searched> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

43) Specifications for Feature X083, “XML namespace declarations in DDL”:

  a) Subclause 12.1, “<column definition>”:

    i) $\boxed{\text{Insert this CR}}$ Without Feature X083, “XML namespace declarations in DDL”, in conforming SQL language, a <generation expression> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

  b) Subclause 12.2, “<check constraint definition>”:

    i) $\boxed{\text{Insert this CR}}$ Without Feature X083, “XML namespace declarations in DDL”, in conforming SQL language, a <check constraint definition> shall not immediately contain and <XML query options> that contains an <XML namespace declaration>.

  c) Subclause 12.4, “<assertion definition>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X083, “XML namespace declarations in DDL”, in conforming SQL language, an <assertion definition> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

44) Specifications for Feature X084, “XML namespace declarations in compound statements”:

a) Subclause 15.1, “<compound statement>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X084, “XML namespace declarations in compound statements”, in conforming SQL language, a <compound statement> shall not immediately contain an <XML query options> that contains an <XML namespace declaration>.

45) Specifications for Feature X090, “XML document predicate”:

a) Subclause 8.2, “<XML document predicate>”:

i) Without Feature X090, “XML document predicate”, conforming SQL language shall not contain <XML document predicate>.

46) Specifications for Feature X100, “Host language support for XML: CONTENT option”:

a) Subclause 12.7, “<SQL-invoked routine>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X100, “Host language support for XML: CONTENT option”, conforming SQL language shall not contain a <document or content> that is CONTENT.

b) Subclause 16.1, “<set XML option statement>”:

i) Without Feature X100, “Host language support for XML: CONTENT option”, conforming SQL language shall not contain a <set XML option statement> that contains CONTENT.

c) Subclause 18.2, “<embedded SQL Ada program>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <Ada XML CLOB variable> shall not immediately contain a <document or content> that is CONTENT.

d) Subclause 18.3, “<embedded SQL C program>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, neither <C XML VARCHAR variable> nor <C XML CLOB variable> shall immediately contain a <document or content> that is CONTENT.

e) Subclause 18.4, “<embedded SQL COBOL program>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <COBOL XML CLOB variable> shall not immediately contain a <document or content> that is CONTENT.

f) Subclause 18.5, “<embedded SQL Fortran program>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <Fortran XML CLOB variable> shall not immediately contain a <document or content> that is CONTENT.

g) Subclause 18.6, “<embedded SQL MUMPS program>”:

i) Insert this CR Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <MUMPS XML CLOB variable> shall not immediately contain a <document or content> that is CONTENT.

h) Subclause 18.7, “<embedded SQL Pascal program>”:

i) Insert this CR Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <Pascal XML CLOB variable> shall not immediately contain a <document or content> that is CONTENT.

i) Subclause 18.8, “<embedded SQL PL/I program>”:

i) Insert this CR Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, neither <PL/I XML VARCHAR variable> nor <PL/I XML CLOB variable> shall immediately contain a <document or content> that is CONTENT.

47) Specifications for Feature X101, “Host language support for XML: DOCUMENT option”:

a) Subclause 12.7, “<SQL-invoked routine>”:

i) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, conforming SQL language shall not contain a <document or content> that is DOCUMENT.

b) Subclause 16.1, “<set XML option statement>”:

i) Without Feature X101, “Host language support for XML: DOCUMENT option”, conforming SQL language shall not contain a <set XML option statement> that contains DOCUMENT.

c) Subclause 18.2, “<embedded SQL Ada program>”:

i) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <Ada XML CLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

d) Subclause 18.3, “<embedded SQL C program>”:

i) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, neither <C XML VARCHAR variable> nor <C XML CLOB variable> shall immediately contain a <document or content> that is DOCUMENT.

e) Subclause 18.4, “<embedded SQL COBOL program>”:

i) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <COBOL XML CLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

f) Subclause 18.5, “<embedded SQL Fortran program>”:

i) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <Fortran XML CLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

g) Subclause 18.6, “<embedded SQL MUMPS program>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <MUMPS XML CLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

h) Subclause 18.7, “<embedded SQL Pascal program>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <Pascal XML CLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

i) Subclause 18.8, “<embedded SQL PL/I program>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, neither <PL/I XML VARCHAR variable> nor <PL/I XML CLOB variable> shall immediately contain a <document or content> that is DOCUMENT.

48) Specifications for Feature X110, “Host language support for XML: VARCHAR mapping”:

a) Subclause 12.7, “<SQL-invoked routine>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X110, “Host language support for XML: VARCHAR mapping”, conforming SQL language shall not contain a <string type option> that contains CHARACTER VARYING, CHAR VARYING, or VARCHAR.

b) Subclause 18.3, “<embedded SQL C program>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X110, “Host language support for XML: VARCHAR mapping”, conforming SQL language shall not contain an <C XML VARCHAR variable>.

c) Subclause 18.8, “<embedded SQL PL/I program>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X110, “Host language support for XML: VARCHAR mapping”, conforming SQL language shall not contain an <PL/I XML VARCHAR variable>.

49) Specifications for Feature X111, “Host language support for XML: CLOB mapping”:

a) Subclause 12.7, “<SQL-invoked routine>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain a <string type option> that contains CHARACTER LARGE OBJECT, CHAR LARGE OBJECT, or CLOB.

b) Subclause 18.2, “<embedded SQL Ada program>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain an <Ada XML CLOB variable>.

c) Subclause 18.3, “<embedded SQL C program>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain an <C XML CLOB variable>.

d) Subclause 18.4, “<embedded SQL COBOL program>”:

i) $\boxed{\text{Insert this CR}}$ Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain an <COBOL XML CLOB variable>.

e) Subclause 18.5, “<embedded SQL Fortran program>”:

  i) Insert this CR Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain an <Fortran XML CLOB variable>.

f) Subclause 18.6, “<embedded SQL MUMPS program>”:

  i) Insert this CR Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain a <MUMPS XML CLOB variable>.

g) Subclause 18.7, “<embedded SQL Pascal program>”:

  i) Insert this CR Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain an <Pascal XML CLOB variable>.

h) Subclause 18.8, “<embedded SQL PL/I program>”:

  i) Insert this CR Without Feature X111, “Host language support for XML: CLOB mapping”, conforming SQL language shall not contain an <PL/I XML CLOB variable>.

50) Specifications for Feature X120, “XML parameters in SQL routines”:

a) Subclause 12.7, “<SQL-invoked routine>”:

  i) Insert this CR Without Feature X120, “XML parameters in SQL routines”, conforming SQL language shall not contain an <SQL-invoked routine> that simply contains a <language clause> that contains SQL and that simply contains a <parameter type> or a <returns data type> that contains a <data type> that is based on either the XML type or a distinct type whose source type is the XML type.

51) Specifications for Feature X121, “XML parameters in external routines”:

a) Subclause 12.7, “<SQL-invoked routine>”:

  i) Insert this CR Without Feature X121, “XML parameters in external routines”, conforming SQL language shall not contain an <SQL-invoked routine> that simply contains a <language clause> that contains ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI and that simply contains a <parameter type> or a <returns data type> that contains a <data type> that is based on either the XML type or a distinct type whose source type is the XML type.

52) Specifications for Feature X131, “Query-level XMLBINARY clause”:

a) Subclause 7.1, “<query expression>”:

  i) Insert this CR Without Feature X131, “Query-level XMLBINARY clause”, in conforming SQL language, a <with clause> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

53) Specifications for Feature X132, “XMLBINARY clause in DML”:

a) Subclause 14.1, “<delete statement: searched>”:

  i) Without Feature X132, “XMLBINARY clause in DML”, in conforming SQL language, a <delete statement: searched> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

b) Subclause 14.2, "<insert statement>":

   i) Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, an <insert statement> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

c) Subclause 14.3, "<merge statement>":

   i) Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, a <merge statement> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

d) Subclause 14.4, "<update statement: positioned>":

   i) Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, an <update statement: positioned> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

e) Subclause 14.5, "<update statement: searched>":

   i) Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, an <update statement: searched> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

54) Specifications for Feature X133, "XMLBINARY clause in DDL":

a) Subclause 12.1, "<column definition>":

   i) Without Feature X133, "XMLBINARY clause in DDL", in conforming SQL language, a <generation expression> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

b) Subclause 12.2, "<check constraint definition>":

   i) Without Feature X133, "XMLBINARY clause in DDL", in conforming SQL language, a <check constraint definition> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

c) Subclause 12.4, "<assertion definition>":

   i) Without Feature X133, "XMLBINARY clause in DDL", in conforming SQL language, an <assertion definition> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

55) Specifications for Feature X134, "XMLBINARY clause in compound statements":

a) Subclause 15.1, "<compound statement>":

   i) Without Feature X134, "XMLBINARY clause in compound statements", in conforming SQL language, a <compound statement> shall not immediately contain an <XML query options> that contains an <XML binary encoding>.

56) Specifications for Feature X135, "XMLBINARY clause in subqueries":

a) Subclause 7.1, "<query expression>":

i) $\boxed{\text{Insert this CR}}$ Without Feature X135, “XMLBINARY clause in subqueries”, in conforming SQL language, a <subquery> shall not contain an <XML query options> that contains an <XML binary encoding>.

Annex B
(informative)
Implementation-defined elements

This Annex modifies Annex B, “Implementation-defined elements”, in ISO/IEC 9075-2.

This Annex references those features that are identified in the body of this part of ISO/IEC 9075 as implementation-defined.

1) Insert this list element Subclause 4.2.1, “Characteristics of XML values”:
   a) Support for the [notations] property and the [unparsed entities] property of the XML root information item is implementation-defined.
2) Subclause 4.8.1, “Mapping SQL character sets to Unicode”:
   a) The mapping of an SQL character set to Unicode is implementation-defined.
3) Subclause 4.8.9, “Mapping Unicode to SQL character sets”:
   a) The mapping of Unicode to a character set in the SQL-environment is implementation-defined.
4) Subclause 4.8.3, “Mapping SQL data types to XML”:
   a) The mapping of numeric SQL types INTEGER, SMALLINT, and BIGINT to XML Schema types is implementation-defined.
5) Subclause 6.10, “<XML element>”:
   a) Whether [notations] and [unparsed entities] are copied from the XML root information item of a nested XML value to the XML root information item of the result is implementation-defined.
   b) The default encoding of binary strings (either in base64 or in hex) is implementation-defined.
6) Subclause 6.11, “<XML forest>”:
   a) Whether [notations] and [unparsed entities] are copied from the XML root information item of a nested XML value to the XML root information item of the result is implementation-defined.
   b) The default encoding of binary strings (either in base64 or in hex) is implementation-defined.
7) Subclause 6.12, “<XML PI>”:
   a) The value of the [base URI] and [notation] properties of the XML processing instruction information item are implementation-defined.
8) Subclause 9.1, “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.

9) Subclause 9.3, “Mapping an SQL table to XML and an XML Schema document”:

   a) The form of the XML declaration is implementation-defined.

   b) The presence and value of the **`schemaLocation`** hint of the **`xsd:import`** in the generated XML Schema document is implementation-defined.

10) Subclause 9.4, “Mapping an SQL schema to an XML document and an XML Schema document”:

   a) The form of the XML declaration is implementation-defined.

   b) The presence and value of the **`schemaLocation`** hint of the **`xsd:import`** in the generated XML Schema document is implementation-defined.

11) Subclause 9.5, “Mapping an SQL catalog to an XML document and an XML Schema document”:

   a) The form of the XML declaration is implementation-defined.

   b) The presence and value of the **`schemaLocation`** hint of the **`xsd:import`** in the generated XML Schema document is implementation-defined.

12) Subclause 9.6, “Mapping an SQL table to XML Schema data types”:

   a) All annotations are implementation-defined.

13) Subclause 9.7, “Mapping an SQL schema to XML Schema data types”:

   a) All annotations are implementation-defined.

14) Subclause 9.8, “Mapping an SQL catalog to XML Schema data types”:

   a) All annotations are implementation-defined.

15) Subclause 9.15, “Mapping SQL data types to XML Schema data types”:

   a) The mapping of numeric SQL types INTEGER, SMALLINT, and BIGINT to XML Schema types is implementation-defined.

   b) All annotations are implementation-defined.

16) Subclause 9.17, “Mapping XML Names to SQL <identifier>s”:

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

17) Subclause 10.13, “Serialization of an XML value”:

   a) The encoding name for any character set other than UTF8 or UTF16 is implementation-defined.

18) Subclause 10.14, “Parsing a character string as an XML value”:

   a) Support for the [notations] property and the [unparsed entities] property of the XML root information item is implementation-defined.

   b) Support for external DTDs is implementation-defined.

19) Subclause 18.3, “<embedded SQL C program>”:

   a) The implicit character set in a <C XML VARCHAR variable>, or a <C XML CLOB variable> is implementation-defined.

20) Subclause 18.4, "<embedded SQL COBOL program>":

   a) The implicit character set in a <COBOL XML CLOB variable> is implementation-defined.

21) Subclause 18.5, "<embedded SQL Fortran program>":

   a) The implicit character set in a <Fortran XML CLOB variable> is implementation-defined.

22) Subclause 18.8, "<embedded SQL PL/I program>":

   a) The implicit character set in a <PL/I XML VARCHAR variable>, or a <PL/I XML CLOB variable> is implementation-defined.

This page intentionally left blank.

Annex C

(informative)

Implementation-dependent elements

This Annex modifies Annex C, “Implementation-dependent elements”, in ISO/IEC 9075-2.

This Annex references those features that are identified in the body of this part of ISO/IEC 9075 as implementation-dependent.

1) Subclause 4.8.7, “Mapping an SQL schema to XML”:
   a) The repeatable ordering of tables in an SQL schema is implementation-dependent.
2) Subclause 6.10, “<XML element>”:
   a) The [base URI] property of the XML element information item is implementation-dependent.
3) Subclause 6.11, “<XML forest>”:
   a) The [base URI] property of each XML element information item is implementation-dependent.
4) Subclause 9.9, “Mapping an SQL data type to an XML Name”:
   a) It is implementation-dependent whether the types of two sites of row type, having the same number of fields, and having corresponding fields of the same name and declared type, are mapped to the same XML Name.
5) Subclause 9.16, “Mapping values of SQL data types to values of XML Schema data types”:
   a) The ordering of elements of a multiset is implementation-dependent.
6) Subclause 10.13, “Serialization of an XML value”:
   a) The result is implementation-dependent, but shall be such that reparsing with XMLPARSE with the same choice of DOCUMENT or CONTENT specified in <XML serialize>, and with the PRESERVE WHITESPACE option, will yield a value identical to the original value.
7) Subclause 10.14, “Parsing a character string as an XML value”:
   a) The [base URI] property of all XML information items is implementation-dependent.
8) Subclause 11.2, “<aggregate function>”:
   a) The order in which items are concatenated in the result of XMLAGG is implementation-dependent if no user-specified ordering is specified or if the user-specified ordering is not a total ordering.

This page intentionally left blank.

Annex D

(informative)

Incompatibilities with ISO/IEC 9075:1999

This Annex modifies Annex E, “Incompatibilities with ISO/IEC 9075:1999”, in ISO/IEC 9075-2.

This edition of this part of ISO/IEC 9075 introduces some incompatibilities with the earlier version of Database Language SQL as specified in ISO/IEC 9075-2:1999.

Except as specified in this Annex, features and capabilities of Database Language SQL are compatible with ISO/IEC 9075-2:1999.

1) A number of additional <reserved word>s have been added to the language. These <reserved word>s are:
   - XMLBINARY
   - XMLCOMMENT
   - XMLPI

This page intentionally left blank.

Annex E

(informative)

SQL feature taxonomy

This Annex describes a taxonomy of features defined in this part of ISO/IEC 9075.

Table 14, “Feature taxonomy for optional features”, contains a taxonomy of the optional features of the SQL language that are specified in this part of ISO/IEC 9075. In this table, the first column contains a counter that may be used to quickly locate rows of the table; these values otherwise have no use and are not stable — that is, they are subject to change in future editions of or even Technical Corrigenda to ISO/IEC 9075 without notice.

The “Feature ID” column of this table specifies the formal identification of each feature and each subfeature contained in the table.

The “Feature Name” column of this table contains a brief description of the feature or subfeature associated with the Feature ID value.

Table 14 — Feature taxonomy for optional features

	Feature ID	Feature Name
1	**X010**	**XML type**
2	**X011**	**Arrays of XML type**
3	**X012**	**Multisets of XML type**
4	**X013**	**Distinct types of XML**
5	**X014**	**Attributes of XML type**
6	**X015**	**Fields of XML type**
7	**X016**	**Persistent XML values**
8	**X020**	**XML Concatenation**
9	**X031**	**XMLElement**
10	**X032**	**XMLForest**
11	**X033**	**XMLRoot**

	Feature ID	Feature Name
12	X034	XMLAgg
13	X035	XMLAgg: ORDER BY option
14	X041	Basic table mapping: null absent
15	X042	Basic table mapping: null as nil
16	X043	Basic table mapping: table as forest
17	X044	Basic table mapping: table as element
18	X045	Basic table mapping: with target namespace
19	X046	Basic table mapping: data mapping
20	X047	Basic table mapping: metadata mapping
21	X048	Basic table mapping: base64 encoding of binary strings
22	X049	Basic table mapping: hex encoding of binary strings
23	X051	Advanced table mapping: null absent
24	X052	Advanced table mapping: null as nil
25	X053	Advanced table mapping: table as forest
26	X054	Advanced table mapping: table as element
27	X055	Advanced table mapping: with target namespace
28	X056	Advanced table mapping: data mapping
29	X057	Advanced table mapping: metadata mapping
30	X058	Advanced table mapping: base64 encoding of binary strings
31	X059	Advanced table mapping: hex encoding of binary strings
32	X060	XMLParse: CONTENT option
33	X061	XMLParse: DOCUMENT option
34	X062	XMLParse: explicit WHITESPACE option
35	X070	XMLSerialize: CONTENT option

	Feature ID	Feature Name
36	X071	XMLSerialize: DOCUMENT option
37	X080	Namespaces in XML publishing
38	X081	Query-level XML namespace declarations
39	X082	XML namespace declarations in DML
40	X083	XML namespace declarations in DDL
41	X084	XML namespace declarations in compound statements
42	X090	XML document predicate
43	X100	Host language support for XML: CONTENT option
44	X101	Host language support for XML: DOCUMENT option
45	X110	Host language support for XML: VARCHAR mapping
46	X111	Host language support for XML: CLOB mapping
47	X120	XML parameters in SQL routines
48	X121	XML parameters in external routines
49	X131	Query-level XMLBINARY clause
50	X132	XMLBINARY clause in DML
51	X133	XMLBINARY clause in DDL
52	X134	XMLBINARY clause in compound statements
53	X135	XMLBINARY clause in subqueries

Table 14, “Feature taxonomy for optional features”, does not provide definitions of the features; the definition of those features is found in the Conformance Rules that are further summarized in Annex A, “SQL Conformance Summary”.

This page intentionally left blank.

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:

$\boxed{\textbf{999}}$ 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**".

Minor Problems and Wordsmithing Candidates: |The following Possible Problem has been noted:

Severity: Minor Technical
Reference: P14, SQL/XML, Subclause 4.8.6, "Mapping an SQL table to XML", et seq
Note at: None.
Source: WG3:DRS-114
Possible Problem:

The schema import together with the xmlns:sqlxml=ˆSQLXMLNS˜ attribute in Subclauses 7.3, 7.4 and 7.5 general rule 2) h) in [SQL/XML WD] can be made implementation-dependent since the imported SQL/XML schema is only used by the implementation-dependent annotations and an implementation that does not use the annotations does not need to import or refer to the SQL/XML schema or namespace.

Outline of possible Solution:

1. Change the Subclause 7.3/4/5 general rule 2) i) introduced by DRS-070R2 to be: i) Let it be implementation-dependent whether XSDIMP is the zero-length string or <xsd:import namespace="SQLXMLNS" schemaLocation="SQLXMLNS.xsd" />

2. Make the presence of the xmlns:sqlxml="SQLXMLNS" declaration in the xsd:schema elements in Subclause 7.3/4/5 general rule 2) j) introduced by DRS-070R2 implementation-dependent.

3. Adjust the Annex C on implementation-dependent elements
Proposed Solution:

None provided with comment.

Language Opportunities: XML-015 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, Clause 9, "Mappings"
Note at: None.
Source: WG3:ZSH-153R1 = H2-2002-153R1
Language Opportunity:

Applications consuming XML parser output are allowed to drop whitespace only content. Since a SQL value of only whitespace characters can be considered preservable information, a mapping option should be provided to add an xml:space="preserve" attribute on the element enclosing string typed values.

An approach to a solution should:

1. Provide an option to specify if the xml:space="preserve" attribute should be added to elements that contain string values.
2. If the option is set, the generated schema will also have to include the schema for the namespace associated with the prefix "xml".

Proposed Solution:

None provided with comment.

XML-011 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, No particular location
Note at: None.
Source: WG3:YYJ-054/H2-2001-_ __
Language Opportunity:

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-017 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, Subclause 6.9, "<XML element>"
Note at: The end of the Format
Source: WG3:ZSH-144R2 = H2-2003-_ __
Language Opportunity:

<XML element> and <XML forest> should have options so that the user can conveniently specify other kinds of null handling. The user should be able to specify whether to represent nulls by absence, by empty content, or by using the xsi:nil="true" attribute.
Proposed Solution:

None provided with comment.

XML-018 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, Clause 9, "Mappings"
Note at: None.
Source: WG3:ZSH-144R2 = H2-2003-_ __
Language Opportunity:

The names of some of the subclauses in this clause are confusing. For example, 7.11 is "Mapping an SQL data type to a named XML schema data type" and 7.15 is "Mapping SQL data types to XML schema data types".
Proposed Solution:

None provided with comment.

XML-019 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, Subclause 9.11, "Mapping an SQL data type to a named XML Schema data type"
Note at: GR 1) ("Let *D* be the SQL data type or domain...").
Source: WG3:ZSH-144R2 = H2-2003-_ __
Language Opportunity:

The following data types have no mappings: structured types, and reference types. Note that both metadata and data mappings are required.
Proposed Solution:

None provided with comment.

XML-020 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, Clause 17, "Dynamic SQL"
Note at: None.
Source: WG3:ZSH-144R2 = H2-2003-_ __
Language Opportunity:

We might want to consider whether to enhance <prepare statement> to support the new functions and operators in Clause 6 of this part. (Due to the restructuring of <prepare statement> in SQL:1999 this is no longer strictly necessary, but if we do not intend to enhance <prepare

statement> it might be better to do so explicitly, say by a paper closing this comment with no action, rather than by failing to act.)
Proposed Solution:

None provided with comment.

XML-021 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, No specific location
Note at: None.
Source: WG3:ZSH-144R2 = H2-2003-_ __
Language Opportunity:

Changes equivalent to the ones made in Subclause 10.2, Data type correspondences, of this part are needed for Subclause 5.15, "SQL/CLI data type correspondences", in Part 3, SQL/CLI FCD document. Also, Table 6, "Codes used for implementation data types in SQL/CLI", Table 7, "Codes used for application data types in SQL/CLI", and Table 37, "Codes used for concise data types", need to be enhanced with a code for XML type.
Proposed Solution:

None provided with comment.

XML-022 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, No specific location
Note at: None.
Source: WG3:ZSH-144R2 = H2-2003-_ __
Language Opportunity:

The ability to store XML Schema documents directly in the database should be provided. These documents may be necessary for validating XML documents.
Proposed Solution:

None provided with comment.

XML-023 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, No specific location
Note at: None.
Source: WG3:ZSH-144R2 = H2-2003-_ __
Language Opportunity:

The ability to validate an XML value against XML schemata is required. This might take the form of a new IS VALID predicate.
Proposed Solution:

None provided with comment.

XML-024 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, Subclause 9.9, "Mapping an SQL data type to an XML Name"
Note at: None.
Source: WG3:ZSH-064 = H2-2003-049
Language Opportunity:

The type name generation does not account for collation information in the context of string types. Further discussion on how to reflect collation in the type mappings are in order. Note: this impacts all places where Subclause 7.9 is referenced, such as in Subclause 7.15, General Rule 8), subrules n), o), p), etc.
Proposed Solution:

None provided with comment.

XML-025 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, Subclause 9.9, "Mapping an SQL data type to an XML Name"
Note at: None.
Source: WG3:ZSH-064 = H2-2003-049
Language Opportunity:

The type name generation does add dynamic SQL type facet information such as the SQL type's precision or string length to the type name. This means that an approach where the predefined SQL types are not defined inline but instead at a global schema leads to a global schema that may contain a very large number of named types that are hard to manage. For example, if both CHAR and VARCHAR allow up to 8000 characters, then that schema contains alone 16,000 XML Schema type definitions. There should be a single SQL/XML XML Schema and associated namespace per implementation (the sqlxml namespace unfortunately cannot be used) to provide a minimal set of mapped base types without this type explosion.
Proposed Solution:

None provided with comment.

XML-026 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, No particular location
Note at: None.
Source: WG3:YYJ-054/H2-2001-_ __ and WG3:ZSH-065 = H2-2003-051
Language Opportunity:

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.

Currently, the mappings provide one approach to map SQL schemata and catalogs using nested elements for scoping. Since this is a fairly new area and different mappings may be devised that are a better fit for some application scenarios (such as using namespaces to identify schemata and catalogs). If and when this LO is addressed, different schema and catalog mapping options will be available that map the schema and catalogs to namespaces instead of to nested XML elements.
Proposed Solution:

None provided with comment.

XML-027 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, No specific location
Note at: None.
Source: WG3:ZSH-063 = H2-2003-048
Language Opportunity:

Currently, the SQL/XML mappings provide an option to define a target namespace that covers the table, both schema and tables, or catalog, schemata, and tables, depending on which mapping is applied. This gives the same target namespace for all three levels. For some approaches, this suffices; however, some applications may want to assign different namespaces to the catalog, the schema, and the table levels.

If and when this LO is addressed, an additional option to allow the assignment of different schema namespace URIs to catalogs and schemata and tables will be needed. It may also be desirable to provide a URI format as the standard format for identifying such components.
Proposed Solution:

None provided with comment.

XML-028 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, No specific location
Note at: None.
Source: WG3:ZSH-083R2 = H2-2003-064R2
Language Opportunity:

The necessary support for accessing values of XML type via SQL/CLI interface needs to be added.
Proposed Solution:

None provided with comment.

XML-029 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, No specific location
Note at: None.
Source: WG3:ZSH-083R2 = H2-2003-064R2
Language Opportunity:

The necessary support for accessing values of XML type via SQL/OLB interface needs to be added.
Proposed Solution:

None provided with comment.

XML-030 The following Language Opportunity has been noted:

Severity: Language Opportunity
Reference: P14, SQL/XML, No specific location
Note at: None.
Source: WG3:ZSH-083R2 = H2-2003-064R2
Language Opportunity:

The necessary support for parameters of XML type for SQL-invoked routines written in Java needs to be added.

Proposed Solution:

None provided with comment.