Final Committee Draft ISO/IEC FCD 9075-14	
Date: **2004-07-23**	Reference number: ISO/JTC 1/SC 32**N1156**
Supersedes document SC 32N1022	

THIS DOCUMENT IS STILL UNDER STUDY AND SUBJECT TO CHANGE. IT SHOULD NOT BE USED FOR REFERENCE PURPOSES.

ISO/IEC JTC 1/SC 32 Data Management and Interchange Secretariat: USA (ANSI)	Circulated to P- and O-members, and to technical committees and organizations in liaison for voting (P-members only) by: **2004-12-06** Please return all votes and comments in electronic form directly to the SC 32 Secretariat by the due date indicated.

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

Title: Information technology — Database Language SQL - Part 14: SQL/XML (for SQL:200n)

Project: 1.32.03.05.14.00

Introductory note: The attached document is hereby submitted for a four-month letter ballot to the National Bodies of ISO/IEC JTC 1/SC 32. The ballot starts 2004-08-06; ITTF must have the document two weeks before the balloting can start.

Medium: E

No. of pages: 360

Address Reply to: Douglas Mann, Secretariat, ISO/IEC JTC 1/SC 32, Pacific Northwest National Laboratory, 13667 Legacy Circle Apt H, Herndon, VA, 20171, United States of America

Telephone: +1 202-566-2126; Facsimile; +1 202-566-1639; E-mail: MannD@battelle.org

ISO/IEC JTC 1/SC 32 N1165

ISO/IEC JTC 1/SC 32

Date: 2005-07-21

FCD ISO/IEC 9075-14:2005 (E)

ISO/IEC JTC 1/SC 32/WG 3

The Netherlands (NNI)

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: Final Committee Draft (FCD)
Document stage: (3) FCD under Consideration
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 9075:2003”, 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.

(Blank page)

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.

(Blank page)

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 9075-1:2003, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*.

[Foundation] ISO/IEC 9075-2:2003, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*.

[Schemata] ISO/IEC 9075-11:2003, *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 1.0] *Extensible Markup Language (XML) Version 1.0 (third edition)*, 4 February, 2004
`http://www.w3.org/TR/REC-xml`

[XML 1.1] *Extensible Markup Language (XML) Version 1.1*, 4 February, 2004

http://www.w3.org/TR/xml11

[XML] is used to reference either [XML 1.0] or [XML 1.1] when there is no significant difference between the two for the purposes of a given citation.

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

[Namespaces 1.0] *Namespaces in XML*, 14 January, 1999
http://www.w3.org/TR/REC-xml-names

[Namespaces 1.1] *Namespaces in XML 1.1*, 4 February, 2004
http://www.w3.org/TR/xml-names11

[Namespaces] is used to reference either [Namespaces 1.0] or [Namespaces 1.1] when there is no significant difference between the two for the purposes of a given citation.

[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

[XQuery] *(Working Draft) XQuery 1.0: an XML Query Language*, W3C working draft, 12 November 2003
http://www.w3.org/TR/xquery/

[XQueryDM] *(Working Draft) XQuery 1.0 and XPath 2.0 Data Model*, W3C working draft, 12 November 2003
http://www.w3.org/TR/xpath-datamodel/

[XQueryFO] *(Working Draft) XQuery 1.0 and XPath 2.0 Functions and Operators*, W3C working draft, 12 November 2003
http://www.w3.org/TR/xpath-functions/

[XQueryFS] *(Working Draft) XQuery 1.0 and XPath 2.0 Formal Semantics*, W3C working draft, 12 November 2003
http://www.w3.org/TR/xquery-semantics/

[Serialization] *(Working Draft) XSLT 2.0 and XQuery 1.0 Serialization*, W3C working draft, 12 November 2003
http://www.w3.org/TR/xslt-xquery-serialization/

[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

(Blank page)

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 XML declaration

3.1.1.4 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

3.1.2.4 wildcard schema component

3.1.2.5 XML Schema

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 empty XQuery sequence: An empty sequence, as defined in [XQueryDM].

3.1.3.4 global element declaration schema component: An element declaration schema component of a global element declaration, as defined in [Schema1].

3.1.3.5 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.

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

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

3.1.3.8 valid XML 1.0 character: A legal character as defined in [XML 1.0], rule [2], “Char”.

3.1.3.9 valid XML 1.1 character: A legal character as defined in [XML 1.1], rule [2], “Char”.

3.1.3.10 XML attribute: An attribute as defined by [XML].

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

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

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

3.1.3.14 XML element: An element as defined by [XML 1.0] or [XML 1.1].

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

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

3.1.3.17 XML 1.0 Name: A Name as defined by [XML 1.0].

3.1.3.18 XML 1.1 Name: A Name as defined by [XML 1.1].

3.1.3.19 XML 1.0 NameChar: A NameChar as defined by [XML 1.0].

3.1.3.20 XML 1.1 NameChar: A NameChar as defined by [XML 1.1].

3.1.3.21 XML 1.1 NameStartChar: A NameStartChar as defined by [XML 1.1].

3.1.3.22 XML namespace: An XML namespace as defined by [Namespaces].

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

3.1.3.24 XML 1.0 NCName: An NCName, as defined by rule [4] in [Namespaces 1.0].

3.1.3.25 XML 1.1 NCName: An NCName, as defined by rule [4] in [Namespaces 1.1].

3.1.3.26 XML 1.0 QName: A QName, as defined by rule [6] of [Namespaces 1.0].

3.1.3.27 XML 1.1 QName: A QName, as defined by rule [6] of [Namespaces 1.1].

3.1.3.28 XML 1.0 QName prefix: A Prefix, as defined by rule [7] of [Namespaces 1.0].

3.1.3.29 XML 1.1 QName prefix: A Prefix, as defined by rule [7] of [Namespaces 1.1].

3.1.3.30 XML 1.0 QName local part: A LocalPart, as defined by rule [8] of [Namespaces 1.0].

3.1.3.31 XML 1.1 QName local part: A LocalPart, as defined by rule [8] of [Namespaces 1.1].

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

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

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

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

3.1.3.36 XML Schema primitive type: A primitive type, as defined in [Schema2].

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

3.1.3.38 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.

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

3.1.3.40 XML target namespace URI: The URI of an XML target namespace.

3.1.3.41 XML text: A character string that is a substring of a textual XML 1.0 content or a textual XML 1.1 content, as defined in Subclause 10.15, "Parsing a string as an XML value".

3.1.3.42 XQuery accessor: An accessor, as defined in [XQueryDM].

3.1.3.43 XQuery atomic type: An atomic type, as defined in [XQueryDM].

3.1.3.44 XQuery atomic value: An atomic value, as defined in [XQueryDM].

3.1.3.45 XQuery attribute node: An attribute node, as defined in [XQueryDM].

3.1.3.46 XQuery comment node: A comment node, as defined in [XQueryDM].

3.1.3.47 XQuery datetime normalized value: The normalized value of a value of XML Schema types xs:dateTime, xs:date, xs:time, or any type derived from these types, as this term is used in [XQueryFO], and implicitly defined in [XQueryDM], section 3.3.3, "Storing xs:dateTime, xs:date and xs:time values in the data model".

3.1.3.48 XQuery datetime timezone component: The timezone component of a value of XML Schema types xs:dateTime, xs:date, xs:time, or any type derived from these types, as this term is used in [XQueryFO] and implicitly defined in [XQueryDM], section 3.3.3, "Storing xs:dateTime, xs:date and xs:time values in the data model".

3.1.3.49 XQuery document node: A document node, as defined in [XQueryDM].

3.1.3.50 XQuery dynamic context: A dynamic context, as defined in [XQuery].

3.1.3.51 XQuery element node: An element node, as defined in [XQueryDM].

3.1.3.52 XQuery error: A static error, type error or dynamic error, as defined in [XQuery].

3.1.3.53 XQuery evaluation with XML 1.0 lexical rules: The process of determining the value of an XQuery expression with XML 1.0 lexical rules, as defined in [XQuery].

3.1.3.54 XQuery evaluation with XML 1.1 lexical rules: The process of determining the value of an XQuery expression with XML 1.1 lexical rules, as defined in [XQuery].

3.1.3.55 XQuery expression context: An expression context, consisting of a static context and a dynamic context, as defined in [XQuery].

3.1.3.56 XQuery expression with XML 1.0 lexical rules: An expression, as defined by rule [40], "Expr", in [XQuery], with rules [19], "NCName", and [21], "QName", interpreted to reference [Namespaces 1.0].

3.1.3.57 XQuery expression with XML 1.1 lexical rules: An expression, as defined by rule [40], "Expr", in [XQuery], with rules [19], "NCName", and [21], "QName", interpreted to reference [Namespaces 1.1].

3.1.3.58 XQuery formal type notation: A formal type notation, as defined by rule [28 (Formal)] "Type", in [XQueryFS].

3.1.3.59 XQuery item: An item, as defined in [XQueryDM].

3.1.3.60 XQuery namespace node: A namespace node, as defined in [XQueryDM].

3.1.3.61 XQuery node: A node, as defined in [XQueryDM].

3.1.3.62 XQuery node identity: Node identity, as defined in [XQueryDM].

3.1.3.63 XQuery processing instruction node: A processing instruction node, as defined in [XQueryDM].

3.1.3.64 XQuery sequence: A sequence, as defined in [XQueryDM].

3.1.3.65 XQuery serialization normalization: Normalization, as defined in [Serialization].

3.1.3.66 XQuery static context: A static context, as defined in [XQuery].

3.1.3.67 XQuery text node: A text node, as defined in [XQueryDM].

3.1.3.68 XQuery tree: A tree, as defined in [XQueryDM].

3.1.3.69 XQuery variable: A variable, as defined in [XQuery].

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.

(Blank page)

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 also defines predefined data types named by the following <key word>: XML.

Augment the 2nd paragraph

— The data types XML(UNTYPED DOCUMENT), XML(ANY DOCUMENT), XML(UNTYPED CONTENT), XML(ANY CONTENT), and XML(SEQUENCE) are referred to as the *XML types*. Values of XML types 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 list in the 4th paragraph

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

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).

— The <XML type modifier> (UNTYPED DOCUMENT, ANY DOCUMENT, UNTYPED CONTENT, ANY CONTENT, or SEQUENCE)

4.2.1 Characteristics of XML values

An *XML value* is either the null value or an XQuery sequence.

Every XML value is a value of type XML(SEQUENCE).

Every XML value that is either the null value or an XQuery document node is a value of type XML(ANY CONTENT).

Every XML value that is either the null value or a non-null value of type XML(ANY CONTENT) that is an XQuery document node *D* such that, for every XQuery element node that is contained in the XQuery tree *T* rooted in *D*, the type property is "**`xdt:untypedAny`**" and the nilled property is "false", and for every XQuery attribute node that is contained in *T*, the type property is "**`xdt:untypedAtomic`**" is a value of type XML(UNTYPED CONTENT).

Every XML value that is either the null value or a non-null value of type XML(ANY CONTENT) that is an XQuery document node whose children property has exactly one XQuery element node and possibly other XQuery nodes permitted as children of an XQuery document node is a value of type XML(ANY DOCUMENT).

Every XML value that is either the null value or a non-null value of type XML(UNTYPED CONTENT) that is an XQuery document node whose children property has exactly one XQuery element node and possibly other XQuery nodes permitted as children of an XQuery document node is a value of type XML(UNTYPED DOCUMENT)

If Feature X211, "XML 1.1 support", is supported, then the value of any property *P* of any XQuery node within an XML value such that *P* is of XQuery atomic type **`xs:QName`** shall be an XML 1.1 QName; otherwise, the value of any such property shall be an XML 1.0 QName.

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 content> 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 XQuery element node 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 6.10, "<XML concatenation>".

<XML comment> is an operator that returns an XML value, given a character string *CS*. The XML value consists of an XQuery comment node whose content property is *CS*, mapped to Unicode. This XQuery comment node may optionally be placed as the sole child of an XQuery document node.

<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 XQuery processing node whose target property is the <identifier>, and whose content property is *CS*, trimmed of leading blanks and mapped to Unicode. This XQuery processing instruction node may optionally be placed as the sole child of an XQuery document node.

<XML query> is an operator that evaluates an XQuery expression, which may be parameterized with any number of input parameters. The result, an XQuery sequence, may optionally be placed in an XQuery document node.

<XML table> is a kind of <derived table>, which may be used to query an XML value as a table.

4.2.4 Registered XML Schemas

A *registered XML Schema* is an XML Schema that has been made known to the SQL-server. The means by which an XML Schema is registered is implementation-defined.

A global element declaration schema component of a registered XML Schema is *non-deterministic* if it contains or references a wildcard schema component whose {namespace constraint} property is either "*not*" together with a namespace name, or "*any*", and whose {process contents} property is either "*strict*" or "*lax*" (as defined in [Schema1] section 3.10.1, "The Wildcard Schema Component").

NOTE 1 — The elements of an XML Schema Document corresponding to such wildcard schema components are elements **`<xs:any>`** or **`<xs:anyAttribute>`** in which the **`namespace`** attribute is either missing or has the value "**`##any`**" or "**`##other`**", and the **`processContents`** attribute is either missing or has either the value "**`strict`**" or the value "**`lax`**".

A registered XML Schema is *non-deterministic* if it contains a global element declaration schema component that is non-deterministic.

A registered XML Schema is described by a registered XML Schema descriptor. A registered XML Schema descriptor includes:

— The target namespace URI of the registered XML Schema.

— The schema location URI of the registered XML Schema.

— The <registered XML Schema name> of the registered XML Schema.

— An indication, whether the registered XML Schema is non-deterministic.

— An indication, whether the registered XML Schema is permanently registered.

— An unordered collection of the namespaces defined by the registered XML Schema (the target namespace is one of these namespaces).

— For each namespace defined by the registered XML Schema, an unordered collection of the global element declaration schema components in that namespace, with an indication for each global element declaration schema component whether that global element declaration schema component is non-deterministic.

It is implementation-defined whether a registered XML Schema is identified by its target namespace URI, or by the combination of its target namespace URI and its schema location URI. A registered XML Schema is also identified by its <registered XML Schema name>.

Certain XML Schemas, defined by either this part of ISO/IEC 9075 or by some normative reference, are always registered. These XML Schemas are enumerated in Table 1, "Permanently registered XML Schemas".

Table 1 — Permanently registered XML Schemas

Common prefix (non- normative)	target namespace URI	Schema location URI	<registered XML Schema name>
`xs`	http://www.w3.org/2001/XMLSchema	implementa-tion-defined	implementa-tion-defined
`xsi`	http://www.w3.org/2001/XMLSchema-instance	implementa-tion-defined	implementa-tion-defined
`sqlxml`	http://standards.iso.org/iso/9075/2003/sqlxml	implementa-tion-defined	implementa-tion-defined

NOTE 2 — The "common prefix" column in the preceding table indicates the prefix(es) commonly associated with the target namespaces of these XML Schemas, but there is no requirement to refer to them by these prefixes.

If an <XML valid predicate> *XVP* that contains an <XML valid according to clause> is contained in a <query expression> of a view, a check constraint, or an assertion, the <triggered action> of a trigger, or in an <SQL-invoked routine>, then the registered XML Schema *RXS* that is referenced by the <XML valid according to what> is determined at the time the view is created, the check constraint is defined, the assertion is created, the trigger is created, or the SQL-invoked routine is created. *RXS* is used to evaluate *XVP* whenever the view is used, the check constraint or assertion is evaluated, the trigger is executed, or the SQL-invoked routine is invoked.

4.3 Data conversions

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

Insert before 3rd paragraph Data conversions between the predefined data types defined in ISO/IEC 9075-2 and the XML types can be specified by an <XML cast specification>.

4.4 Data analysis operations (involving tables)

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

4.4.1 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 formed from the <XML value expression> evaluated for each row that qualifies.

4.5 SQL-invoked routines

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

4.5.1 Routine descriptors

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

Augment the routine descriptor of SQL routines

— For every SQL parameter whose declared type is an XML type or a distinct type whose source type is an XML type, an indication of the <XML passing mechanism>.

— If the SQL routine is an SQL-invoked function, then an indication of the <XML passing mechanism> of the <returns clause>.

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.

4.6 SQL-statements

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

4.6.1 SQL-statements classified by function

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

4.6.1.1 SQL-session statements

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

Insert this paragraph The following are additional SQL-session statements:

— <set XML option statement>

4.7 Basic security model

This Subclause modifies Subclause 4.34, "Basic security model", in ISO/IEC 9075-2.

4.7.1 Privileges

This Subclause modifies Subclause 4.34.2, "Privileges", in ISO/IEC 9075-2.

Augment the list following 1st paragraph

— registered XML schema

Augment the list following 8th paragraph

— registered XML Schema

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

4.8 SQL-sessions

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

4.8.1 SQL-session properties

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

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

Insert at the end of dash list in 12nd paragraph

— The current XML option.

4.9 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 2, "XML namespace prefixes and their URIs".

Table 2 — 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
xdt	http://www.w3.org/2003/11/xpath-datatypes
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.

4.10 Overview of mappings

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

— Mapping SQL character sets to Unicode.

— Mapping SQL <identifier>s to XML Names.

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

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

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

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

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

The mappings from XML to SQL include:

— Mapping Unicode to SQL character sets.

— Mapping XML Names to SQL <identifier>s.

4.10.1 Mapping SQL character sets to Unicode

For each character set *SQLCS* in the SQL-environment, there shall be a mapping *CSM* of strings of *SQLCS* to strings of Unicode. 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 **`<`**, **`&`**, **`>`**, **`&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.

4.10.2 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.

NOTE 4 — This part of this International Standard specifies no syntax for invoking the partially escaped mapping specified in Subclause 9.1, "Mapping SQL <identifier>s to XML Names". This specification is intended to be used by applications and referenced by other standards.

4.10.3 Mapping SQL data types to XML

For each SQL type or domain, with the exception of structured types and reference types, 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, distinct type, or domain *SQLT* is mapped to the XML Schema type ***`XMLT`*** that is the closest analog to *SQLT*. Since the value space of ***`XMLT`*** is frequently richer than the set of values that

can be represented by *SQLT*, facets are used to restrict ***XMLT*** in order to capture the restrictions on *SQLT* as much as possible.

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

The SQL character string types are mapped to the XML Schema type **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 displace-

ment, 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 **`xdt:yearMonthDuration`** and **`xdt: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.

4.10.4 Mapping values of SQL data types to XML

For each SQL type or domain *SQLT*, with the exception of structured types and reference types, there is also a mapping of values of type *SQLT* to the value space of the corresponding XML Schema type. The mappings of values are largely determined by the data type mappings. The precise rules for non-null values are found in Subclause 9.16, “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"`**.

4.10.5 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, or XML(SEQUENCE). 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 *C* is a visible column of *T* for authorization identifier *U* and the declared type of *C* is not an XML unmappable data type.

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

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

4.10.6 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 5 — 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.

4.10.7 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 6 — 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.

4.10.8 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 7 — 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.

4.10.9 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*.

4.10.10 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 8 — 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 9 — The sequence of mappings from SQL <identifier> to XML Name (using either the fully escaped mapping or the partially escaped mapping) to SQL <identifier> restores the original SQL <identifier> (assuming that every character in the source SQL-implementation's SQL <identifier> is a character of SQL_TEXT in the target SQL-implementation). However, the sequence of mappings from XML Name to SQL <identifier> to XML Name does not necessarily restore the XML Name. Also, more than one XML Name may be mapped to the same SQL <identifier>.

4.11 Mapping XQuery atomic values to SQL values

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

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

Case:

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

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

— If ***AT*** is **xdt:dayTimeDuration**, or derived from **xdt:dayTimeDuration**, then **xdt:dayTime-Duration**.

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

This part of ISO/IEC 9075 (notably, in Subclause 6.3, "<cast specification>", regards ***AV*** as being a value belonging to some category of SQL predefined type, as follows:

Case:

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

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

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

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

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

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

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

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

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

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

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

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

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

  | ABSENT | ACCORDING

  | BASE64

  | COLUMNS | CONTENT

  | DOCUMENT

  | EMPTY | ENCODING

  | HEX

  | ID

  | LOCATION

  | NAMESPACE | NIL

  | PASSING | PATH | PRESERVE

  | RETURNING

  | SEQUENCE | STANDALONE | STRIP

  | UNTYPED | URI

  | VALID | VERSION

  | WHITESPACE
```

```
  | XMLSCHEMA

<reserved word> ::=

    !! All alternatives from ISO/IEC 9075-2

  | XML | XMLAGG | XMLATTRIBUTES | XMLBINARY | XMLCAST
  | XMLCONCAT | XMLCOMMENT | XMLELEMENT | XMLEXISTS | XMLFOREST
  | XMLITERATE | XMLNAMESPACES | XMLPARSE | XMLPI
  | XMLQUERY | XMLSERIALIZE | XMLTABLE
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

5.2 Names and identifiers

This Subclause modifies Subclause 5.4, "Names and identifiers", in ISO/IEC 9075-2.

Function

Specify names.

Format

```
<registered XML Schema name> ::= <schema qualified name>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR A <registered XML Schema name> identifies a registered XML Schema.

Conformance Rules

No additional Conformance Rules.

5.3 <value expression primary>

This Subclause modifies Subclause 6.3, "<value expression primary>", in ISO/IEC 9075-2.

Function

Specify a value that is syntactically self-delimited.

Format

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

    !! All alternatives from ISO/IEC 9075-2
  | <XML cast specification>
```

Syntax Rules

1) Replace SR 1) The declared type of a <value expression primary> is the declared type of the simply contained <value expression>, <unsigned value specification>, <column reference>, <set function specification>, <window function>, <scalar subquery>, <case expression>, <cast specification>, <XML cast specification>, <field reference>, <subtype treatment>, <method invocation>, <static method invocation>, <new specification>, <attribute or method reference>, <reference resolution>, <collection value constructor>, <array element reference>, <multiset element reference>, or <next value expression>, or the effective returns type of the simply contained <routine invocation>, respectively.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 1) The value of a <value expression primary> is the value of the simply contained <value expression>, <unsigned value specification>, <column reference>, <set function specification>, <window function>, <scalar subquery>, <case expression>, <cast specification>, <XML cast specification>, <field reference>, <subtype treatment>, <method invocation>, <static method invocation>, <new specification>, <attribute or method reference>, <reference resolution>, <collection value constructor>, <array element reference>, <multiset element reference>, <routine invocation>, or <next value expression>, respectively.

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 [ <left paren> <XML type modifier> <right paren> ]

<XML type modifier> ::=
    [ <XML untyped or any> ] DOCUMENT
  | [ <XML untyped or any> ] CONTENT
  | SEQUENCE

<XML untyped or any> ::= UNTYPED | ANY
```

Syntax Rules

1) Insert this SR XML specifies an XML data type.

2) Insert this SR If an <XML type> does not specify <XML type modifier>, then it is implementation-defined whether ANY CONTENT or UNTYPED CONTENT is implicit; otherwise, if <XML untyped or any> is not specified, then it is implementation-defined whether UNTYPED or ANY is implicit.

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) and the explicit or implicit <XML type modifier>.

Conformance Rules

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

2) 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 an XML type or a distinct type whose source type is an XML type.

3) 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 an XML type or a distinct type whose source type is an XML type.

4) Insert this CR Without Feature X181, “XML(UNTYPED DOCUMENT) type”, conforming SQL language shall not contain an <XML type> whose <XML type modifier> is UNTYPED DOCUMENT.

5) Insert this CR Without Feature X182, “XML(ANY DOCUMENT) type”, conforming SQL language shall not contain an <XML type> whose <XML type modifier> is ANY DOCUMENT.

6) Insert this CR Without Feature X231, “XML(UNTYPED CONTENT) type”, conforming SQL language shall not contain an <XML type> whose <XML type modifier> is UNTYPED CONTENT.

7) Insert this CR Without Feature X232, “XML(ANY CONTENT) type”, conforming SQL language shall not contain an <XML type> whose <XML type modifier> is ANY CONTENT.

8) Insert this CR Without Feature X190, “XML(SEQUENCE) type”, conforming SQL language shall not contain an <XML type> whose <XML type modifier> is SEQUENCE.

6.2 <field definition>

This Subclause modifies Subclause 6.2, "<field definition>", in ISO/IEC 9075-2.

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) 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 an XML type or a distinct type whose source type is an 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 5) If the <cast operand> specifies an <empty specification>, then *TD* shall be a collection type, XML(UNTYPED CONTENT), XML(ANY CONTENT), or XML(SEQUENCE).

2) 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:

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

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


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

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 2)b) If the <cast operand> specifies an <empty specification>, then

   Case:

   a) If *TD* is XML(UNTYPED CONTENT) or XML(ANY CONTENT), then the result of *CS* is an XQuery document node whose children and unparsed-entity properties are empty, and whose base-uri and document-uri properties are implementation-defined. No further General Rules of this Subclause are applied.

   b) If *TD* is XML(SEQUENCE), then the result of *CS* is an empty XQuery sequence, and no further General Rules of this Subclause are applied.

   c) Otherwise, the result of *CS* is an empty collection of declared type *TD*, and no further General Rules of this Subclause are applied.

2) Insert before GR 3) If *SD* is an XML type, then:

   a) Case:

      i) If *TD* is XML(UNTYPED DOCUMENT) or XML(ANY DOCUMENT), and *SV* is not a value of type XML(ANY DOCUMENT), then an exception is raised: *data exception — not an XML document*.

      ii) If *TD* is XML(UNTYPED CONTENT) or XML(ANY CONTENT), and *SV* is not a value of type XML(ANY CONTENT), then an exception is raised: *data exception — not an XQuery document node*

   b) Case:

      i) If *TD* is XML(UNTYPED DOCUMENT) or XML(UNTYPED CONTENT), then let *TV* be the result of applying the General Rules of Subclause 10.18, "Constructing an unvalidated XQuery document node", to *SV*.

      ii) Otherwise, *TV* is *SV*.

Conformance Rules

No additional Conformance Rules.

6.4 <XML cast specification>

Function

Specify a data conversion whose source or target type is an XML type.

Format

```
<XML cast specification> ::=
    XMLCAST <left paren> <XML cast operand> AS
    <XML cast target> <right paren>

<XML cast operand> ::=
    <value expression>
  | <implicitly typed value specification>

<XML cast target> ::=
    <domain name>
  | <data type>
```

Syntax Rules

1) Case:

   a) If the <XML cast specification> 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 cast specification>.

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

2) Case:

   a) If *XBE* is 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) Case:

   a) If <XML cast target> is <domain name>, then let *TD* be the data type of the specified domain.

   b) Otherwise, let *TD* be the data type identified by <data type>.

4) At least one of the following shall be true:

   a) *TD* is an XML type.

   b) The <XML cast operand> is a <value expression> whose declared type is an XML type.

5) *TD* shall not be a collection type, a row type, a structured type or a reference type.

6) The declared type of the result of the <XML cast specification> is *TD*.

7) If *TD* is XML(UNTYPED DOCUMENT) or XML(ANY DOCUMENT), then the <XML cast operand> shall be either NULL or a <value expression> whose declared type is an XML type.

8) If the <XML cast operand> is a <value expression> VE, then let *SD* be the declared type of the <value expression>.

   a) *SD* shall not be a collection type, a row type, a structured type or a reference type.

   b) If *SD* is an XML type, and *TD* is an XML type, then the <XML cast specification> is equivalent to

      ```
      CAST ( VE AS TD )
      ```

9) If <XML cast operand> is an <implicitly typed value specification> *ITVS*, then the <XML cast specification> is equivalent to

   ```
   CAST ( ITVS AS TD )
   ```

10) If *TD* is character string type, then the descriptor of *TD* shall not contain <collate clause>. The declared type collation of the <XML cast specification> is the character set collation of the character set of *TD* and its collation derivation is implicit.

11) If <XML cast target> is <domain name>, then let *D* be the domain identified by the <domain name>. The schema identified by the explicit or implicit qualifier of the <domain name> shall include the descriptor of *D*.

Access Rules

1) If *TD* is a distinct type, then let *STD* be the source type of *TD*, and let *AVE* be an arbitrary <value expression> of declared type *STD*. The Access Rules of Subclause 6.12, "<cast specification>", in ISO/IEC 9075-2 are applied to

   ```
   CAST ( VE AS TD )
   ```

2) If *SD* is a distinct type, then let *SSD* be the source type of *SD*. The Access Rules of Subclause 6.12, "<cast specification>", in ISO/IEC 9075-2 are applied to

   ```
   CAST ( AVE AS SSD )
   ```

3) If <XML cast target> is a <domain name>, then

   Case:

   a) If the <XML cast specification> is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, then the applicable privileges of the <authorization identifier> that owns the containing SQL schema shall include USAGE on the domain identified by the <domain name>.

   b) Otherwise, the current privileges shall include USAGE on the domain identified by the <domain name>.

General Rules

1) Let *V* be the value of the <XML cast operand> simply contained in the <XML cast specification>.

2) If *V* is the null value, then the result of the <XML cast specification> is the null value, and no further General Rules of this Subclause are evaluated.

3) If *TD* is an XML type, then:

a) Let *XMLV* be a <character string literal> whose value is the result of evaluating the General Rules of Subclause 9.16, "Mapping values of SQL data types to values of XML Schema data types", with *V* as the value of an SQL data type, *ENC* as *ENCODING*, "absent" as *NULLS*, and *True* as *CHARMAPPING*.

b) Let *TEMPV* be result of

```
XMLPARSE (CONTENT XMLV PRESERVE WHITESPACE)
```

c) If *TD* is XML(UNTYPED CONTENT) or XML(ANY CONTENT), then let *TV* be *TEMPV*.

d) If *TD* is XML(SEQUENCE), then:

i) Let *XST* be the XML Schema type obtained by applying the General Rules of Subclause 9.15, "Mapping SQL data types to XML Schema data types", with *SD* as the SQL data type, *ENC* as the *ENCODING*, and "absent" as *NULLS*.

ii) Case:

1) If *SD* is a year-month interval type, then let ***XSBT*** be the XQuery simple type **xdt:yearMonthDuration**.

2) If *SD* is a day-time interval type, then let ***XSBT*** be the XQuery simple type **xdt:dayTimeDuration**.

3) If *XST* is an XML Schema built-in type, then let *XSBT* be *XST*.

4) If *XST* an XML Schema atomic type, then let *XSBT* be the XML Schema built-in type from which *XST* is derived.

iii) Let ***XSBTN*** be an XML 1.1 QName for *XSBT*.

iv) Let *XSC* be an XQuery static context created according to the Syntax Rules of Subclause 10.19, "Creation of an XQuery expression context", with the *BNF* non-terminal argument omitted. Let *XDC* be an XQuery dynamic context created according to the General Rules of Subclause 10.19, "Creation of an XQuery expression context".

v) Let *XSC* and *XDC* be augmented with an XQuery variable whose XML 1.1 QName is **TEMP**, whose XQuery formal type notation is "**document { text ? }**", and whose value is *TEMPV*.

vi) Let *TV* be the result of the XQuery evaluation with XML 1.1 lexical rules, using *XSC* and *XDC* as the XQuery expression context, of the XQuery expression

```
$TEMP cast as XSBTN
```

If this XQuery evaluation raises an XQuery error, then an exception condition is raised: *XQuery error*.

e) *TV* is the result of the <XML cast specification>.

4) If *SD* is an XML type and *TD* is not an XML type, then:

a) Case:

i) If the <XML cast target> is a <domain name>, then let *SQLT* be the <data type> of the identified domain.

ii) If If the <XML cast target> identifies a distinct type, then let *SQLT* be the source type of the distinct type.

iii) Otherwise, let *SQLT* be <data type>.

b) Let *XV* be the result of applying the General Rules of Subclause 10.16, "Removing XQuery document nodes from an XQuery sequence", with *V* as the *SEQUENCE*.

c) Let *AV* be the result of applying the XQuery function **`fn:data()`** to *XV*.

d) If *AV* is the empty sequence, then the result is the null value and no further General Rules of this Subclause are executed.

e) Let *XT* be the XML Schema type obtained by applying the General Rules of Subclause 9.15, "Mapping SQL data types to XML Schema data types", with *SQLT* as the SQL data type, *ENC* as the *ENCODING*, and "absent" as *NULLS*.

f) Let ***`XMLT`*** be a XML 1.1 QName for *XT*.

NOTE 10 — ***`XMLT`*** may be in one of the built-in namespaces denoted by the prefixes **`xs:`** or **`xdt:`**, or it may be in an implementation-dependent namespace, not necessarily available to the user.

g) Let *XSC* be an XQuery static context created according to the Syntax Rules of Subclause 10.19, "Creation of an XQuery expression context", with the *BNF* non-terminal argument omitted.

h) Let *XDC* be an XQuery dynamic context created according to the General Rules of Subclause 10.19, "Creation of an XQuery expression context".

i) Let *XSC* and *XDC* be augmented with an XQuery variable whose QName is **`TEMP`**, whose XQuery formal type notation is "**`xdt:untypedAtomic`**", and whose value is *AV*.

j) Let *BV* be the result of the XQuery evaluation with XML 1.1 lexical rules, using *XSC* and *XDC* as the XQuery expression context, of the XQuery expression

```
$TEMP cast as XMLT
```

If this XQuery evaluation raises an XQuery error, then an exception condition is raised: *XQuery error*.

k) *BV* is an XQuery atomic value.

NOTE 11 — The following rules assume that the value space of XQuery atomic types is the same as the value space of corresponding SQL types. That is, it is possible to treat *BV* as a value of an SQL type. For example, if *BV* is of type **`xs:integer`**, then *BV* is a (mathematical) integer, and, as such, it may also be the value of an exact numeric type with scale 0 (zero). See Subclause 4.11, "Mapping XQuery atomic values to SQL values".

Case:

i) If ***`XT`*** is **`xs:string`** or derived from **`xs:string`**, then let *A* be an arbitrary <literal> of character string type whose character repertoire is Unicode and whose value is *BV*.

ii) If ***`XT`*** is **`xs:hexBinary`** or **`xs:base64Binary`**, or derived from **`xs:hexBinary`** or **`xs:base64Binary`**, then let *A* be an arbitrary <literal> of binary string type whose value is *BV*.

iii) If ***`XT`*** is **`xs:decimal`** or derived from **`xs:decimal`**, then let *A* be an arbitrary <literal> of exact numeric type whose value is *BV*.

iv) If ***`XT`*** is **`xs:float`** or **`xs:double`**, or derived from **`xs:float`** or **`xs:double`**, then:

1) If *AV* is `INF`, `-INF`, or `NaN`, then an exception condition is raised: *data exception — numeric value out of range*.

2) Let *A* be an arbitrary <literal> of approximate numeric type whose value is *BV*.

v) If ***XT*** is **`xs:date`** or derived from **`xs:date`**, then:

1) If the XQuery datetime normalized value component of *AV* is not positive, or if the XQuery datetime timezone component of *AV* is not the XQuery empty sequence, then an exception condition is raised: *data exception — invalid datetime format*.

2) Let *A* be an arbitrary <literal> of declared type *SQLT* whose value is *BV* .

vi) If ***XT*** is **`xs:dateTime`** or derived from **`xs:dateTime`**, then:

1) If the XQuery datetime normalized value component of *AV* is not positive, then an exception condition is raised: *data exception — invalid datetime format*.

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

A) If the XQuery datetime timezone component of *AV* is not the XQuery empty sequence, then an exception condition is raised: *data exception — invalid datetime format*.

B) Let *A* be an arbitrary <literal> of declared type TIMESTAMP WITHOUT TIME ZONE whose value is *BV*.

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

A) If the XQuery datetime timezone component of *AV* is the XQuery empty sequence, then an exception condition is raised: *data exception — invalid datetime format*.

B) Let *A* be an arbitrary <literal> of declared type TIMESTAMP WITH TIME ZONE whose value is *BV*.

vii) If ***XT*** is **`xs:time`** or derived from **`xs:time`**, then

Case:

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

A) If the XQuery datetime timezone component of *AV* is not the XQuery empty sequence, then an exception condition is raised: *data exception — invalid datetime format*.

B) Let *A* be an arbitrary <literal> of declared type TIME WITHOUT TIME ZONE whose value is the XQuery datetime normalized value of *BV*.

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

A) If the XQuery datetime timezone component of *AV* is the XQuery empty sequence, then an exception condition is raised: *data exception — invalid datetime format*.

B) Let *A* be an arbitrary <literal> of declared type TIME WITH TIME ZONE whose value is *BV* (more precisely, whose UTC component is the XQuery datetime normalized value of *BV* and whose timezone component is the XQuery datetime timezone component of *BV*).

viii) If ***XT*** is **`xdt:yearMonthDuration`** or derived from **`xdt:yearMonthDuration`**, then let *A* be an arbitrary <literal> of year-month interval type whose value is *BV*.

ix) If ***XT*** is **xdt:dayTimeDuration** or derived from **xdt:dayTimeDuration**, then let *A* be an arbitrary <literal> of day-time interval type whose value is *BV*.

x) If ***XT*** is **xs:boolean** or derived from **xs:boolean**, then let *A* be an arbitrary <literal> of type BOOLEAN whose value is *BV*.

l) The result of the <XML cast specification> is the result of

```
CAST (A AS SQLT)
```

NOTE 12 — This raises an exception condition if the value of *A* does not conform to constraints of *SQLT*. It may also perform rounding or truncation, and so forth, in order to produce a value of type *SQLT*.

Conformance Rules

None.

6.5 <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>.

3) Insert after SR 7)s) An <XML valid predicate> *XVP* that satisfies one of the following:

   a) *XVP* does not specify <XML valid according to clause>.

   b) *XVP* specifies an <XML valid according to clause> that identifies a nondeterministic registered XML Schema and *XVP* does not specify an <XML valid element clause>.

   c) *XVP* specifies an <XML valid element clause> that identifies a nondeterministic global element declaration schema component of a registered XML Schema.

NOTE 13 — This implies that an <XML valid predicate> that is used in a <check constraint definition> or an <assertion definition> shall contain an <XML valid according to clause> that either identifies a deterministic registered XML Schema, or contains an <XML valid element clause> that identifies a deterministic global element declaration schema component of a registered XML Schema.

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.

6.6 <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 character string serialization>

<XML character string serialization> ::=
    XMLSERIALIZE <left paren> <document or content or sequence>
    <XML value expression> AS <data type>
    [ <XML serialize version> ] <right paren>

<document or content or sequence> ::=
    <document or content>
  | SEQUENCE

<document or content> ::=
    DOCUMENT
  | CONTENT

<XML serialize version> ::= VERSION <character string literal>

<blob value function> ::=
    !! All alternatives for ISO/IEC 9075-2
  | <XML binary string serialization>

<XML binary string serialization> ::=
    XMLSERIALIZE <left paren> <document or content or sequence>
    <XML value expression> AS <data type>
    [ ENCODING <XML encoding specification> ]
    [ <XML serialize version> ] <right paren>

<XML encoding specification> ::= <XML encoding name>

<XML encoding name> ::= <SQL language identifier>
```

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 character string serialization>.

2) Insert this SR If <XML character string serialization> is specified, then:

a) The <data type> shall be a character string type. The declared type of <XML character string serialization> is <data type>.

b) The <character string literal> immediately contained in <XML serialize version> shall be '1.0' or '1.1', or it shall identify some successor to [XML 1.0] and [XML 1.1].

c) If <XML serialize version> is not specified, then an implementation-defined <XML serialize version> is implicit.

3) Replace SR 15) The declared type of <blob value function> is the declared type of the immediately contained <blob substring function>, <blob trim function>, <blob overlay function>, or <XML binary string serialization>.

4) Insert this SR If <XML binary string serialization> is specified, then:

a) The <data type> shall be a binary string type.

b) The declared type of <XML binary string serialization> is <data type>.

c) If <XML encoding specification> is not specified, then an implementation-defined <XML encoding name> is implicit.

d) The supported <XML encoding name>s are implementation-defined.

e) The <character string literal> immediately contained in <XML serialize version> shall be '1.0' or '1.1', or it shall identify some successor to [XML 1.0] and [XML 1.1].

f) If <XML serialize version> is not specified, then an implementation-defined <XML serialize version> is implied.

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 character string serialization>.

2) Insert this GR If <XML character string serialization> is specified, then let *DCS* be the <document or content or sequence>, let *XMLV* be the value of the <XML value expression>, let *DT* be the <data type>, let *CS* be the character set of *DT*, and let *VER* be the <string value expression> simply contained in the explicit or implicit <XML serialize version>. The result of <XML character string serialization> is the result determined by applying the General Rules of Subclause 10.14, "Serialization of an XML value", with *DCS* as *SYNTAX*, *XMLV* as *VALUE*, *DT* as *TYPE*, *CS* as *ENCODING*, and *VER* as *VERSION*.

3) Replace GR 11) The result of <blob value function> is the result of the simply contained <blob substring function>, <blob trim function>, <blob overlay function>, or <XML binary string serialization>.

4) Insert this GR If <XML binary string serialization> is specified, then let *DCS* be the <document or content or sequence>, let *XMLV* be the value of the <XML value expression>, let *DT* be the <data type>, let *XEN* be the <XML encoding name>, and let *VER* be the <string value expression> simply contained in the

explicit or implicit <XML serialize version>. The result of <XML binary string serialization> is the result determined by applying the General Rules of Subclause 10.14, "Serialization of an XML value", with *DCS* as *SYNTAX*, *XMLV* as *VALUE*, *DT* as *TYPE*, *XEN* as *ENCODING*, and *VER* as *VERSION*.

Conformance Rules

1) Insert this CR Without Feature X070, "XMLSerialize: Character string serialization and CONTENT option", conforming SQL language shall not contain an <XML character string serialization> that immediately contains a <document or content or sequence> that is CONTENT.

2) Insert this CR Without Feature X071, "XMLSerialize: Character string serialization and DOCUMENT option", conforming SQL language shall not contain an <XML character string serialization> that immediately contains a <document or content or sequence> that is DOCUMENT.

3) Insert this CR Without Feature X072, "XMLSerialize: Character string serialization and SEQUENCE option", conforming SQL language shall not contain an <XML character string serialization> that immediately contains a <document or content or sequence> that is SEQUENCE.

4) Insert this CR Without Feature X073, "XMLSerialize: BLOB serialization and CONTENT option", conforming SQL language shall not contain an <XML binary string serialization> that immediately contains a <document or content or sequence> that is CONTENT.

5) Insert this CR Without Feature X074, "XMLSerialize: BLOB serialization and DOCUMENT option", conforming SQL language shall not contain an <XML binary string serialization> that immediately contains a <document or content or sequence> that is DOCUMENT.

6) Insert this CR Without Feature X075, "XMLSerialize: BLOB serialization and SEQUENCE option", conforming SQL language shall not contain an <XML binary string serialization> that immediately contains a <document or content or sequence> that is SEQUENCE.

7) Insert this CR Without Feature X076, "XMLSerialize: VERSION", in conforming SQL language, <XML character string serialization> shall not contain VERSION.

8) Insert this CR Without Feature X076, "XMLSerialize: VERSION", in conforming SQL language, <XML binary string serialization> shall not contain VERSION.

6.7 <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 an XML type.

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

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>.

6.8 <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 query>
```

Syntax Rules

1) The declared type of <XML value function> is the declared type of the simply contained <XML comment>, <XML concatenation>, <XML element>, <XML forest>, <XML parse>, <XML PI>, or <XML query>.

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 query>.

Conformance Rules

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

6.9 <XML comment>

Function

Generate an XML value with a single XQuery comment node, possibly as a child of an XQuery document node.

Format

```
<XML comment> ::=
    XMLCOMMENT <left paren> <string value expression>
    [ <XML returning clause> ] <right paren>
```

Syntax Rules

1) If <XML returning clause> is not specified, then RETURNING CONTENT is implicit.

2) Case:

   a) If RETURNING CONTENT is specified or implied, then it is implementation-defined whether the declared type of <XML comment> is XML(UNTYPED CONTENT) or XML(ANY CONTENT).

   b) Otherwise, the declared type of <XML comment> is XML(SEQUENCE).

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 XQuery comment node 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 property is initially set to 'empty'.

         3) Case:

A) If RETURNING CONTENT is specified or implied, then:

I) Let *XRII* be an XQuery document node with the following properties:

1) The base-uri property is implementation-defined.

2) The children property is a list with one element, *XCII*.

3) The unparsed entities property is the empty set.

4) The document-uri property is implementation-defined.

II) The parent property of *XCII* is set to *XRII*.

III) The result of the <XML comment> is the XQuery sequence consisting of one node, *XRII*.

B) Otherwise, the result of the <XML comment> is the XQuery sequence consisting of one node, *XCII*.

Conformance Rules

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

2) Without Feature X241, "Explicit RETURNING CONTENT in XML publishing", in Conforming SQL language, an <XML comment> shall not specify an <XML returning clause> that is RETURNING CONTENT.

3) Without Feature X242, "RETURNING SEQUENCE in XML publishing", in Conforming SQL language, an <XML comment> shall not specify an <XML returning clause> that is RETURNING SEQUENCE.

6.10 <XML concatenation>

Function

Concatenate a list of XML values.

Format

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

Syntax Rules

1) If <XML returning clause> is not specified, then RETURNING CONTENT is implicit.

2) The declared type of <XML concatenation> is

   Case:

   a) If RETURNING CONTENT is specified or implied, then

      Case:

      i) If the declared type of any <XML value expression> simply contained in the <XML concatenation> is XML(ANY DOCUMENT), XML(ANY CONTENT), or XML(SEQUENCE), then XML(ANY CONTENT).

      ii) Otherwise, XML(UNTYPED CONTENT).

   b) Otherwise, XML(SEQUENCE).

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) Let *XRC* be the explicit or implicit <XML returning clause>.

4) For all *i*, 1 (one) ≤ *i* ≤ *n*, let R_i be the result of applying the General Rules of Subclause 10.13, "Concatenation of two XML values", with R_{i-1} and XV_i as the XML values and *XRC* as the *RETURNING*.

5) 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>.

2) Without Feature X241, “Explicit RETURNING CONTENT in XML publishing”, in Conforming SQL language, an <XML concatenation> shall not specify an <XML returning clause> that is RETURNING CONTENT.

3) Without Feature X242, “RETURNING SEQUENCE in XML publishing”, in Conforming SQL language, an <XML concatenation> shall not specify an <XML returning clause> that is RETURNING SEQUENCE.

6.11 <XML element>

Function

Generate an XML value with a single XQuery element node, possibly as a child of an XQuery document node.

Format

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

<XML content option> ::=
    NULL ON NULL
  | EMPTY ON NULL
  | ABSENT ON NULL
  | NIL ON NULL
  | NIL ON NO CONTENT
```

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 predefined type other than XML, or a distinct type whose source type is not an XML type.

   d) Case:

      i) If A_i contains an <XML attribute name> AN_i, then let ANC_i be 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 1.1 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 10.10, "Determination of namespace URI", 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, $NSURI_i$ is identical as defined in [Namespaces] to $NSURI_j$, 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>. *EN* shall be an XML 1.1 QName.

6) The Syntax Rules of Subclause 10.10, "Determination of namespace URI", 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 collection type that is not based on an XML unmappable type, or a distinct type.

8) If <XML element content> is specified, and <XML content option> is not specified, then EMPTY ON NULL is implicit.

9) If <XML element content> is specified, then there shall not be an <XML attribute> A_i such that $NSURI_i$ is the XML namespace URI given in Table 2, "XML namespace prefixes and their URIs", corresponding to the XML namespace prefix **xsi**, and the XML QName local part of ANC_i is **nil**.

10) If <XML returning clause> is not specified, then RETURNING CONTENT is implicit.

11) Case:

a) If RETURNING CONTENT is specified or implied, then it is implementation-defined whether the declared type of <XML element> is XML(UNTYPED CONTENT) or XML(ANY CONTENT).

b) Otherwise, the declared type of <XML element> is XML(SEQUENCE).

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) 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>.

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) 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, *ENC* as the *ENCODING*, "absent" as *NULLS*, and *False* as *CHARMAPPING*, resulting in a character string CAV_i of Unicode characters.

b) Let AII_i be an XQuery attribute node having the following properties:

i) The node name is an XQuery atomic value of XQuery atomic type **`xs:QName`**, whose local name is the XML 1.1 Qname local part of ANC_i and whose namespace URI is $NSURI_i$.

ii) The string-value property is CAV_i.

iii) The type property is **`xdt:untypedAtomic`**.

iv) The parent property is initially unknown.

4) Let *ATTRS* be the XQuery sequence of XML attribute nodes AII_i, 1 (one) ≤ *i* ≤ *n*, such that AV_i is not null.

5) Let *N* be the number of <XML element content>s. Let V_j be an enumeration of the values of the <XML element content>s, 1 (one) ≤ *j* ≤ *N*. Let *CON* be the list of values V_1, ... , V_N.

6) The result of <XML element> is the obtained by applying the General Rules of Subclause 10.11, "Construction of an XML element", with *EN* as the *QNAME*, *ENSURI* as the *NAMESPACE*, *ATTRS* as the *ATTRIBUTES*, *CON* as the *CONTENTS*, the explicit or implicit <XML content option> as the *OPTION*, *ENC* as the *ENCODING*, and the explicit or implicit <XML returning clause> as the *RETURNING*.

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>.

3) Without Feature X170, "XML null handling options", conforming SQL language shall not specify <XML content option>.

4) Without Feature X171, "NIL ON NO CONTENT option", conforming SQL language shall not specify NIL ON NO CONTENT.

5) Without Feature X211, "XML 1.1 support", in conforming SQL language, an <XML element name> shall be an XML 1.0 QName.

6) Without Feature X211, “XML 1.1 support”, in conforming SQL language, an <XML attribute name> shall be an XML 1.0 QName.

7) Without Feature X241, “Explicit RETURNING CONTENT in XML publishing”, in Conforming SQL language, an <XML element> shall not specify an <XML returning clause> that is RETURNING CONTENT.

8) Without Feature X242, “RETURNING SEQUENCE in XML publishing”, in Conforming SQL language, an <XML element> shall not specify an <XML returning clause> that is RETURNING SEQUENCE.

6.12 <XML forest>

Function

Generate an XML value with a list of XQuery element nodes, possibly as the children of an XQuery document node.

Format

```
<XML forest> ::=
    XMLFOREST <left paren> [ <XML namespace declaration> <comma> ]
    <forest element list>
    [ OPTION <XML content option> ]
    [ <XML returning clause> ]
    <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 collection type that is not based on an XML unmappable type, or a distinct type.

   c) Case:

      i) If F_i contains a <forest element name> FEN_i, then let FNC_i be 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 1.1 QName.

   e) The Syntax Rules of Subclause 10.10, "Determination of namespace URI", are applied, with FNC_i as *QNAME* and with the <XML forest> as *BNFTERM*, resulting in an XML namespace URI $NSURI_i$.

3) If <XML content option> is not specified, then NULL ON NULL is implicit.

4) If <XML returning clause> is not specified, then RETURNING CONTENT is implicit.

5) Case:

   a) If RETURNING CONTENT is explicit or implicit, then it is implementation-defined whether the declared type of <XML forest> is XML(UNTYPED CONTENT) or XML(ANY CONTENT).

   b) Otherwise, the declared type of <XML element> is XML(SEQUENCE).

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) For each *i* between 1 (one) and *n*, let CON_i be the list of values consisting of the single value V_i.

5) Let *XRC* be the explicit or implicit <XML returning clause>.

6) For each *i* between 1 (one) and *n*, let XV_i be the result obtained by applying the General Rules of Subclause 10.11, "Construction of an XML element", with FNC_i as the *QNAME*, $NSURI_i$ as the *NAMESPACE*, the empty XQuery sequence as the *ATTRIBUTES*, CON_i as the *CONTENTS*, the explicit or implicit <XML content option> as the *OPTION*, *ENC* as the *ENCODING*, and *XRC* as the *RETURNING*.

7) Let R_1 be XV_1. For all *i*, $2 \le i \le n$, let R_i be the concatenation of R_{i-1} and XV_i, as determined by applying the General Rules of Subclause 10.13, "Concatenation of two XML values", with *XRC* as the *RETURNING*.

8) The result of <XML forest> is R_n.

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>.

3) Without Feature X211, “XML 1.1 support”, in conforming SQL language, a <forest element name> shall be an XML 1.0 QName.

4) Without Feature X241, “Explicit RETURNING CONTENT in XML publishing”, in Conforming SQL language, an <XML forest> shall not specify an <XML returning clause> that is RETURNING CONTENT.

5) Without Feature X242, “RETURNING SEQUENCE in XML publishing”, in Conforming SQL language, an <XML forest> shall not specify an <XML returning clause> that is RETURNING SEQUENCE.

6.13 <XML parse>

Function

Perform a non-validating parse of a 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 the result of <XML parse> is XML.

2) Case:

   a) If <document or content> is DOCUMENT, then it is implementation-defined whether the declared type of <XML parse> is XML(UNTYPED DOCUMENT) or XML(ANY DOCUMENT).

   b) Otherwise, it is implementation-defined whether the declared type of <XML parse> is XML(UNTYPED CONTENT) or XML(ANY CONTENT).

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) Case:

   a) If *V* is the null value, then the result of <XML parse> is the null value.

   b) Otherwise, the result of <XML parse> is determined by applying the General Rules of Subclause 10.15, "Parsing a string as an XML value", with *DC* as *SYNTAX*, *V* as *TEXT*, and *WO* as *OPTION*.

Conformance Rules

1) Without Feature X060, "XMLParse: Character string input and CONTENT option", in conforming SQL language, the declared type of the <string value expression> immediately contained in <XML parse> shall

not be a character string type and <XML parse> shall not immediately contain a <document or content> that is CONTENT.

2) Without Feature X061, "XMLParse: Character string input and DOCUMENT option", in conforming SQL language, the declared type of the <string value expression> immediately contained in <XML parse> shall not be a character string type and <XML parse> shall not immediately contain a <document or content> that is DOCUMENT.

3) Without Feature X065, "XMLParse: BLOB input and CONTENT option", in conforming SQL language, the declared type of the <string value expression> immediately contained in <XML parse> shall not be a binary string type and <XML parse> shall not immediately contain a <document or content> that is CONTENT.

4) Without Feature X066, "XMLParse: BLOB input and DOCUMENT option", in conforming SQL language, the declared type of the <string value expression> immediately contained in <XML parse> shall not be a binary string type and <XML parse> shall not immediately contain a <document or content> that is DOCUMENT.

6.14 <XML PI>

Function

Generate an XML value with a single XQuery processing instruction node, possibly as a child of an XQuery document node.

Format

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

<XML PI target> ::= <identifier>
```

Syntax Rules

1) Let *I* be the result of mapping the <identifier> contained in the <XML PI target> to Unicode.

2) *I* shall be an XML 1.1 Name.

3) *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'.

4) If <XML returning clause> is not specified, then RETURNING CONTENT is implicit.

5) Case:

   a) If RETURNING CONTENT is explicit or implicit, then it is implementation-defined whether the declared type of <XML PI> is XML(UNTYPED CONTENT) or XML(ANY CONTENT).

   b) Otherwise, the declared type of <XML PI> is XML(SEQUENCE).

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) If the value of

'<?' *I* || ' ' || *SVE* || '?>'

does not conform to rule [16], "PI", of [XML 1.1], then an exception condition is raised: *data value — invalid processing instruction*.

NOTE 14 — If the implementation does not support XML 1.1, then the processing instruction target *I* is subject to a Conformance Rule limiting it to be an XML 1.0 Name. Under that assumption, if the proposed processing instruction conforms to rule [16] of [XML 1.1], then it also conforms to rule [16] of [XML 1.0].

iii) Let *XPIII* be an XQuery processing instruction node with the following properties:

1) The value of the target property is *I*.

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 parent property is initially set to 'empty'.

iv) Case:

1) If RETURNING CONTENT is specified or implicit, then:

A) Let *XRII* be an XMQuery document node with the following properties:

I) The base-uri property is implementation-defined.

II) The children property is a list with one element, *XPIII*.

III) The unparsed-entities property is the empty set.

IV) The document-uri property is implementation-defined.

B) The parent property of *XPIII* is set to *XRII*.

C) The result of the <XML PI> is *XRII*.

2) Otherwise, the result of the <XML PI> is *XPIII*.

Conformance Rules

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

2) Without Feature X211, "XML 1.1 support", in conforming SQL language, a <XML PI target> shall be an XML 1.0 QName.

3) Without Feature X241, "Explicit RETURNING CONTENT in XML publishing", in Conforming SQL language, an <XML PI> shall not specify an <XML returning clause> that is RETURNING CONTENT.

4) Without Feature X242, "RETURNING SEQUENCE in XML publishing", in Conforming SQL language, an <XML PI> shall not specify an <XML returning clause> that is RETURNING SEQUENCE.

6.15 <XML query>

Function

Evaluate an XQuery expression.

Format

```
<XML query> ::=
    XMLQUERY <left paren>
    <XQuery expression>
    [ <XML query argument list> ]
    <XML returning clause>
    [ <XML query returning mechanism> ]
    <right paren>

<XQuery expression> ::= <character string literal>

<XML query argument list> ::=
    PASSING <XML query default passing mechanism>
    <XML query argument>
    [ { <comma> <XML query argument> }... ]

<XML query default passing mechanism> ::=
    <XML passing mechanism>

<XML query argument> ::=

    <XML query context item>
  | <XML query variable>

<XML query context item> ::=
    <value expression> [ <XML passing mechanism> ]

<XML query variable> ::=
    <value expression> AS <identifier>
    [ <XML passing mechanism> ]

<XML query returning mechanism> ::=
    <XML passing mechanism>
```

Syntax Rules

1) Let *XMQ* be the <XML query>.

2) If *XMQ* contains <XML query argument list>, then let DPM be the <XML query default passing mechanism> immediately contained in the <XML query argument list>.

3) Case:

   a) If the <XML returning clause> is RETURNING CONTENT, then <XML query returning mechanism> shall not be specified. Let *XQRM* be BY VALUE.

b) If *XMQ* contains <XML query returning mechanism>, then let *XQRM* be that <XML query returning mechanism>.

c) Otherwise, *XMQ* shall contain <XML query argument list>. Let *XQRM* be *DPM*.

4) Let XSC be an XQuery static context defined by applying the Syntax Rules of Subclause 10.19, "Creation of an XQuery expression context", with the <XML query> as the *BNF*, modifying the in-scope variables component of XSC as follows:

a) For each <XML query variable> *XQV*:

i) The <identifier> *I* contained in *XQV* shall be an XML 1.1 NCName.

ii) *I* shall not be equivalent to the name of any implementation-defined XQuery variable, or to the <identifier> of any other <XML query variable>.

iii) Let *VVE* be the <value expression> simply contained in *XQV*.

iv) The declared type of *VVE* shall be convertible to XML(SEQUENCE) according to the Syntax Rules of Subclause 6.4, "<XML cast specification>".

v) If the declared type of *VVE* is not an XML type, then *XQV* shall not contain an <XML passing mechanism>.

vi) Let *VFT* be the XQuery formal type notation determined by the Syntax Rules of Subclause 10.20, "Determination of an XQuery formal type notation", with *VVE* as the *SOURCE*.

vii) The pair (*I*, *VFT*) is placed in the in-scope variables of *XSC*.

b) If *XMQ* contains <XML query context item>, then

i) *XMQ* shall contain exactly one <XML query context item> *XQCI*.

ii) Let *CVE* be the <value expression> simply contained in *XQCI*.

iii) The declared type of *CVE* shall be convertible to XML(SEQUENCE) according to the Syntax Rules of Subclause 6.4, "<XML cast specification>".

iv) If the declared type declared type of *CVE* is not an XML type, then *XQCI* shall not contain an <XML passing mechanism>.

v) If *XQCI* contains an <XML passing mechanism>, then let *CPM* be that <XML passing mechanism>; otherwise, let *CPM* be *DPM*.

vi) Let *CFT* be the XQuery formal type notation determined by the Syntax Rules of Subclause 10.20, "Determination of an XQuery formal type notation", with *CVE* as the *SOURCE*.

vii) Let ***fs*:dot** be the XQuery variable in the fictitious namespace ***fs*:** posited in [XQuery FS], section 3.1.2, "Dynamic context", corresponding to the context item in *XSC*. The pair (***fs*:dot**, *CFT*) is placed in the in-scope variables component of *XSC*.

NOTE 15 — Initialization of the XQuery static context can in many cases be overridden by the prolog of the <XQuery expression>.

5) <XQuery expression> shall conform to the XQuery rules of XQuery expressions that, using the XML 1.1 lexical rules, define the static analysis phase using the XQuery static context *XSC* without raising an XQuery error. It is implementation-defined which optional features of XQuery are supported.

6) Case:

a) If <XML query returning mechanism> contains RETURNING CONTENT, then it is implementation-defined whether the declared type of <XML query> is XML(UNTYPED CONTENT) or XML(ANY CONTENT).

b) Otherwise, the declared type of <XML query> is XML(SEQUENCE).

Access Rules

None.

General Rules

1) Let *XDC* be an XQuery dynamic context created by applying the General Rules of Subclause 10.19, "Creation of an XQuery expression context".

   a) Let *DPM* be the <XML query default passing mechanism>.

   b) Case:

      i) If there is no <XML query context item>, then there is no context item in *XDC*.

      ii) Otherwise, let *CVE* be the <value expression> simply contained in the <XML query context item>.

          Case:

          1) If the value of *CVE* is the null value, then the result of the <XML query> is the null value, and no further General Rules of this Subclause are applied.

          2) Otherwise:

             A) Case:

                I) If the declared type of *CVE* is an XML type, then let *CI* be

                   Case:

                   1) If *CPM* is BY REF, then the value of *CVE*.

                   2) Otherwise, the result of applying the General Rules of Subclause 10.17, "Constructing a copy of an XML value", with the value of *CVE* as the *VALUE*.

                II) Otherwise, let *CI* be the result of the <XML cast specificaton>:

                ```
                XMLCAST (CVE AS XML(SEQUENCE))
                ```

             B) Case:

                I) If *CI* is the empty XQuery sequence, then the result of the <XML query> is the null value, and no further rules of this Subclause are applied.

                II) If *CI* is an XQuery sequence of length 1 (one), then the context item, context position, and context size of *XDC* are set by initializing **fs:dot** to reference *CI*, **fs:position** to 1 (one), and **fs:last** to 1 (one).

III) Otherwise, an exception condition is raised: *data exception — invalid XQuery context item*.

c) For each <XML query variable> *XQV*:

i) Let *VVE* be the <value expression> simply contained in *XQV*, and let *V* be the value of *VVE*.

Case:

1) If *V* is the null value, then let *VV* be the empty sequence.

2) If *V* is a value of an XML type, then let *VV* be

Case:

A) If *VPM* is BY REF, then *V*.

B) Otherwise, the result of applying the General Rules of Subclause 10.17, "Constructing a copy of an XML value", with *V* as the *VALUE*.

3) Otherwise, let *VV* be the result of

```
XMLCAST ( VVE AS XML(SEQUENCE))
```

ii) The XQuery variable whose QName is equivalent to the <identifier> simply contained in *XQV* is set to *VV*.

2) Case:

a) If the implementation supports Feature X211, "XML 1.1 support", then the <XQuery expression> is evaluated as an XQuery expression with XML 1.1 lexical rules, using *XSC* and *XDC* as the XQuery expression context, yielding a value *X1* of an XML type.

b) Otherwise, the <XQuery expression> is evaluated as an XQuery expression with XML 1.0 lexical rules, using *XSC* and *XDC* as the XQuery expression context, yielding a value *X1* of an XML type.

3) If the result of the XQuery evaluation is an XQuery error, then an exception condition is raised: *XQuery error*.

4) Case:

a) If the <XML returning clause> is RETURNING SEQUENCE, then let *X2* be *X1*.

b) Otherwise, let *X2* be the result of XQuery serialization normalization applied to *X1*.

5) Case:

a) If *XQRM* is BY REF, then let *X3* be *X2*.

b) Otherwise, let *X3* be the result of applying the General Rules of Subclause 10.17, "Constructing a copy of an XML value", with *X2* as the *VALUE*.

6) *X3* is the result of the <XML query>.

Conformance Rules

1) Without Feature X201, “XMLQuery: RETURNING CONTENT”, conforming SQL language shall not contain an <XML query> that contains RETURNING CONTENT.

2) Without Feature X202, “XMLQuery: RETURNING SEQUENCE”, conforming SQL language shall not contain an <XML query> that contains RETURNING SEQUENCE.

3) Without Feature X211, “XML 1.1 support”, in conforming SQL language, the value of the <XQuery expression> shall be an XQuery expression with XML 1.0 lexical rules.

4) Without Feature X211, “XML 1.1 support”, in conforming SQL language, the <identifier> contained in an <XML query variable> shall be an XML 1.0 NCName.

(Blank page)

7 Query expressions

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

7.1 <table reference>

This Subclause modifies Subclause 7.6, "<table reference>", in ISO/IEC 9075-2.

Function

Reference a table.

Format

```
<table primary> ::=

    !! All alternatives from ISO?IEC 9075-2
  | <XML iterate> [ AS ] <correlation name>
    <left paren> <derived column list> <right paren>
  | <XML table> [ AS ] <correlation name>
    [ <left paren> <derived column list> <right paren> ]

<XML iterate> ::=
    XMLITERATE <left paren> <XML value expression> <right paren>

<XML table> ::=
    XMLTABLE <left paren>
    [ <XML namespace declaration> <comma> ]
    <XML table row pattern>
    [ <XML table parameters> ]
    COLUMNS <XML table column definitions>

<XML table row pattern> ::= <character string literal>

<XML table parameters> ::=
    PASSING <XML query argument>
    [ { <comma> <XML query argument> }... ]

<XML table column definitions> ::=
    <XML table column definition>
    [ { <comma> <XML table column definition> }... ]

<XML table column definition> ::=

    <XML table ordinality column definition>
  | <XML table regular column definition>
```

```
<XML table ordinality column definition> ::=
    <column name> FOR ORDINALITY

<XML table regular column definition> ::=
    <column name> <data type> [ <XML passing mechanism> ]
    [ <default clause> ]
    [ PATH <XML table column pattern> ]

<XML table column pattern> ::= <character string literal>
```

Syntax Rules

1) Replace SR 5) If a <table factor> *TF* is contained in a <from clause> *FC* with no intervening <query expression>, then the *scope clause SC* of *TF* is the <select statement: single row> or innermost <query specification> that contains *FC*. The scope of the range variable of *TF* is the <select list>, <where clause>, <group by clause>, <having clause>, and <window clause> of *SC*, together with every <lateral derived table> that is simply contained in *FC* and is preceded by *TF*, and every <collection derived table> that is simply contained in *FC* and is preceded by *TF*, and every <XML iterate> that is simply contained in *FC* and is preceded by TF, and the <join condition> of all <joined table>s contained in *SC* that contain *TF*. If *SC* is the <query specification> that is the <query expression body> of a simple table query *STQ*, then the scope of the range variable of *TF* also includes the <order by clause> of *STQ*.

2) Insert this SR If <table primary> contains <XML iterate>, then the <derived column list> shall contain exactly two <column name>s *CN1* and *CN2*, which shall not be equivalent. The declared type of <XML iterate> is a row type consisting of two fields, the first of which has a <field name> that is equivalent to *CN1* and a data type that is XML(SEQUENCE), and the second of which has a <field name> that is equivalent to *CN2* and a data type that is exact numeric with scale 0 (zero).

3) Insert this SR If <XML table> is specified, then:

   a) Case:

      i) If <XML namespace declaration> *XND* is specified, then let *XNDC* be

         `WITH` *XND*

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

   b) Let *XTRP* be the <XML table row pattern>.

   c) Case:

      i) If <XML table parameters> is specified, then let *NA* be the number of <XML query argument>s. Let XQA_i, 1 (one) ≤ *i* ≤ *NA*, be an enumeration of the <XML query argument>s. Let *XQAL* be

         `PASSING BY REF` XQA_1`, ...,` XQA_{NA}

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

   d) Let *NC* be the number of <XML table column definition>s. Let $XTCD_j$, 1 (one) ≤ j ≤ *NC*, be an enumeration of the <XML table column definition>s, in order from left to right. There shall be at most one $XTCD_j$ that is <XML table ordinality column definition>.

   e) For each *j* between 1 (one) and *NC*, inclusive, let CN_j be the <column name> contained in $XTCD_j$.

Case:

i) If $XTCD_j$ is an <XML table ordinality column definition>, then let SLI_j be

```
I.N AS CNj
```

ii) Otherwise:

1) Let DT_j be the <data type> contained in $XTCD_j$.

2) Case:

A) If DT_j is XML(SEQUENCE) then <XML passing mechanism> shall be specified. Let XPM_j be this <XML passing mechanism>. Let XRM_j be RETURNING SEQUENCE XPM_j.

B) Otherwise, <XML passing mechanism> shall not be specified. Let XRM_j be RETURNING CONTENT.

3) If $XTCD_j$ contains a <default option>, then let DEF_j be that <default option>; otherwise, let DEF_j be NULL.

4) If $XTCD_j$ contains an <XML table column pattern>, then let $PATH_j$ be that <XML table column pattern>; otherwise, let $PATH_j$ be CN_j.

5) Let XQC_j be

```
XMLQUERY ( PATHj PASSING BY REF I.V XRMj )
```

6) Let XE_j be

```
XMLEXISTS ( PATHj PASSING BY REF I.V )
```

7) Let SLI_j be

```
CASE WHEN XEj
    THEN CAST( XQCj AS DTj )
    ELSE DEFj
END
```

f) Let *CORR* be the <correlation name>.

g) Case:

i) If <derived column list> is specified, then let *DCL* be that <derived column list> and let *DCLP* be

```
( DCL )
```

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

h) The <XML table> is equivalent to

```
LATERAL
  ( XNDC
    SELECT SLI1, SLI2, ..., SLINC
    FROM XMLITERATE ( XMLQUERY ( XTRP XQAL
                      RETURNING SEQUENCE BY REF ) )
         AS I ( V, N )
  ) AS CORR DCLP
```

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 2) If a <table primary> *TP* simply contains an <XML iterate>, then let *V* be the value of the <XML value expression>.

   Case:

   a) If *V* is the null value or the empty XQuery sequence, then the result of *TP* is an empty table.

   b) Otherwise, the result of *TP* is a table consisting of one row for each XQuery item in *V*. The value of the first column of each row is the corresponding XQuery item in *V*. The value of the second column is the sequential number of the XQuery item in *V*.

Conformance Rules

1) Insert this CR Conforming SQL language shall not contain <XML iterate>.

2) Insert this CR Without Feature X095, “XMLTable”, conforming SQL language shall not contain <XML table>.

7.2 <query expression>

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

Function

Specify a table.

Format

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

Syntax Rules

1) Insert this SR A <with clause> shall contain an <XML lexically scoped option> or a <with list> or both.

2) Insert this SR The scope of an <XML namespace declaration> contained in an <XML lexically scoped 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 lexically scoped 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 lexically scoped 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 lexically scoped 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 lexically scoped 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 lexically scoped options> that contains an <XML binary encoding>.

(Blank page)

8 Predicates

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

8.1 <predicate>

This Subclause 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 content predicate>
  | <XML document predicate>
  | <XML valid predicate>
  | <XML exists 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>, <XML content predicate>, <XML document predicate>, <XML valid predicate>, or <XML exists predicate>.

Conformance Rules

No additional Conformance Rules.

8.2 <XML content predicate>

Function

Determine whether an XML value is a value of type XML(UNTYPED CONTENT) or XML(ANY CONTENT).

Format

```
<XML content predicate> ::=
    <XML value expression> IS [ NOT ]
    [ <XML untyped or any> ] CONTENT
```

Syntax Rules

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

2) If <XML untyped or any> is not specified, then ANY is implicit.

3) Let *UA* be either a zero-length string, UNTYPED, or ANY. The expression

   XVE IS NOT *UA* CONTENT

   is equivalent to

   NOT (*XVE* IS *UA* CONTENT)

Access Rules

None.

General Rules

1) Let *V* be the value of the <XML value expression>.

2) The result of

   XVE IS UNTYPED CONTENT

   is determined as follows.

   Case:

   a) If *V* is the null value, then the result *Unknown*.

   b) If *V* is a value of type XML(UNTYPED CONTENT), then the result is *True*.

   c) Otherwise, the result is *False*.

3) The result of

   XVE IS ANY CONTENT

is determined as follows.

Case:

a) If *V* is the null value, then the result *Unknown*.

b) If *V* is a value of type XML(ANY CONTENT), then the result is *True*.

c) Otherwise, the result is *False*.

Conformance Rules

1) Without Feature X091, "XML content predicate", conforming SQL language shall not contain <XML content predicate>.

2) Without Feature X231, "XML(UNTYPED CONTENT) type", in conforming SQL language, an <XML content predicate> shall not contain <XML untyped or any> that is UNTYPED.

3) Without Feature X232, "XML(ANY CONTENT) type", in conforming SQL language, an <XML content predicate> shall not contain <XML untyped or any> that is ANY.

8.3 <XML document predicate>

Function

Determine whether an XML value is a value of type XML(UNTYPED DOCUMENT) or XML(ANY DOCUMENT). item.

Format

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

Syntax Rules

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

2) If <XML untyped or any> is not specified, then ANY is implicit.

3) Let *UA* be either a zero-length string, UNTYPED, or ANY. The expression

   XVE IS NOT *UA* DOCUMENT

   is equivalent to

   NOT (*XVE* IS *UA* DOCUMENT)

Access Rules

None.

General Rules

1) Let *V* be the value of the <XML value expression>.

2) The result of

   XVE IS UNTYPED DOCUMENT

   is determined as follows.

   Case:

   a) If *V* is the null value, then the result is *Unknown*.

   b) If *V* is a value of type XML(UNTYPED DOCUMENT), then the result is *True*.

   c) Otherwise, the result is *False*.

3) The result of

XVE IS ANY DOCUMENT

is determined as follows.

Case:

a) If *V* is the null value, then the result is *Unknown*.

b) If *V* is a value of type XML(ANY DOCUMENT), then the result is *True*.

c) Otherwise, the result is *False*.

Conformance Rules

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

2) Without Feature X181, "XML(UNTYPED DOCUMENT) type", in conforming SQL language, an <XML document predicate> shall not contain <XML untyped or any> that is UNTYPED.

3) Without Feature X182, "XML(ANY DOCUMENT) type", in conforming SQL language, an <XML document predicate> shall not contain <XML untyped or any> that is ANY.

8.4 <XML exists predicate>

Function

Specify a test for a non-empty XQuery sequence.

Format

```
<XML exists predicate> ::=
    XMLEXISTS <left paren> <XQuery expression>
    [ <XML query argument list> ]<right paren>
```

Syntax Rules

1) Let *XQE* be the <XQuery expression>.

2) If <XML query argument list> is specified, then let *XQP* be that <XML query argument list>; otherwise, let *XQP* be the zero-length string.

3) The Syntax Rules of Subclause 6.15, "<XML query>", are applied to

   `XMLQUERY ( XQE XQP RETURNING SEQUENCE )`

Access Rules

1) The Access Rules of Subclause 6.15, "<XML query>", are applied to

   `XMLQUERY ( XQE XQP RETURNING SEQUENCE )`

General Rules

1) Let *V* be the value of

   `XMLQUERY ( XQE XQP RETURNING SEQUENCE )`

2) The value of <XML exists predicate> is

   Case:

   a) If *V* is the null value, then *Unknown*.

   b) If *V* is an empty XQuery sequence, then *False*.

   c) Otherwise, *True*.

Conformance Rules

1) Without Feature X096, "XMLExists", conforming SQL language shall not contain <XML exists predicate>.

8.5 <XML valid predicate>

Function

Specify a test to determine whether an XML value is valid according to a registered XML Schema. item.

Format

```
<XML valid predicate> ::=
    <XML value expression> IS [ NOT ] VALID
    [ <XML valid identity constraint option> ]
    [ <XML valid according to clause> ]

<XML valid identity constraint option> ::=
    WITHOUT IDENTITY CONSTRAINTS
  | WITH IDENTITY CONSTRAINTS GLOBAL
  | WITH IDENTITY CONSTRAINTS LOCAL
  | DOCUMENT

<XML valid according to clause> ::=
    ACCORDING TO XMLSCHEMA <XML valid according to what>
    [ <XML valid element clause> ]

<XML valid according to what> ::=
    <XML valid according to URI>
  | <XML valid according to identifier>

<XML valid according to URI> ::=
    URI <XML valid target namespace> [ <XML valid schema location> ]
  | NO NAMESPACE [ <XML valid schema location> ]

<XML valid target namespace> ::= <XML URI>

<XML URI> ::= <character string literal>

<XML valid schema location> ::= LOCATION <XML valid schema location URI>

<XML valid schema location URI> ::= <XML URI>

<XML valid according to identifier> ::= ID <registered XML Schema name>

<XML valid element clause> ::=
    [ NAMESPACE <XML valid element namespace> ]
    ELEMENT <XML valid element name>

<XML valid element namespace> ::= <XML URI>

<XML valid element name> ::= <identifier>
```

Syntax Rules

1) Let *XVE* be the <XML value expression>. Let *XVICO* be the <XML valid identity constraint option>, if any; otherwise, let *XVICO* be the zero-length string. Let *XVACC* be the <XML valid according to clause>, if any; otherwise, let *XVACC* be the zero-length string.

2) If <XML valid predicate> immediately contains NOT, then the <XML valid predicate> is equivalent to

```
NOT ( XVE IS VALID XVICO XVACC )
```

3) If <XML valid predicate> does not immediately contain NOT, then

Case:

a) If <XML valid identity constraint option> is not specified, then the <XML valid predicate> is equivalent to

```
( XVE IS VALID WITHOUT IDENTITY CONSTRAINTS XVACC )
```

b) If <XML valid identity constraint option> is specified and is DOCUMENT, then the <XML valid predicate> is equivalent to

```
( ( XVE IS DOCUMENT )
AND
  ( XVE IS VALID
    WITH IDENTITY CONSTRAINTS GLOBAL XVACC ) )
```

NOTE 16 — If <XML valid identity constraint option> is specified and is not DOCUMENT, then there are no further syntactic transformations.

4) If <XML valid according to clause> is specified, then:

a) If <XML valid according to identifier> is specified, then the <registered XML Schema name> shall identify a registered XML Schema.

b) If <XML valid according to URI> is specified, then:

i) Case:

1) If NO NAMESPACE is specified, then let *NSURI* be the zero-length string.

2) Otherwise, let *NSURI* be the <XML valid target namespace>.

ii) Case:

1) If the SQL-implementation identifies registered XML Schemas by the combination of their target namespace URIs and schema location URIs, then

Case:

A) If <XML valid schema location> is specified, then there shall exist a registered XML Schema *SXS* whose target namespace is identical to *NSURI*, as defined by [Namespaces], and whose schema location equals <XML valid schema location> according to the UCS_BASIC collation.

B) Otherwise, a schema location is chosen using a determinstic implementation-defined algorithm that is repeatable, in the sense that if the algorithm is re-evaluated with the same collection of registered XML Schemas that are accessible to the user, the same schema location will be chosen. There shall exist a registered XML Schema *SXS* whose namespace is identical to *NSURI*, as defined by [Namespaces], and whose schema location is as determined by the algorithm.

NOTE 17 — This does not say that the algorithm must choose a schema location from among the schema locations of registered XML Schemas. One possibility for the implementation-defined algorithm is to choose a schema location that has no registererd XML Schema, resulting in a syntax error.

2) Otherwise, there shall exist a registered XML Schema *SXS* whose target namespace is *NSURI*.

c) If <XML valid element clause> is specified, then:

i) Case:

1) If <XML valid element namespace> is specified, then let *ENSURI* be <XML valid element namespace>.

2) Otherwise, let *ENSURI* be *NSURI*.

ii) Let *EN* be the <XML valid element name>. The registered XML Schema *SXS* shall have a global element declaration schema component whose namespace is *ENSURI* and whose XML NCName is *EN*.

Access Rules

1) If <XML valid according to clause> is specified, then

Case:

a) If the <XML valid predicate> is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then the applicable privileges of the <authorization identifier> that owns the containing schema shall include USAGE on *SXS*.

b) Otherwise, the current privileges shall include USAGE on *SXS*.

General Rules

1) Let *V* be the value of *XVE*.

2) Case:

a) If *V* is the null value, then the result of the <XML valid predicate> is *Unknown*.

b) If *V* does not contain exactly one XQuery item, then the result of the <XML valid predicate> is *False*.

c) If *V* is not an XQuery document node or an XQuery element node, then the result of the <XML valid predicate> is *False*.

d) If *V* is an XQuery document node and the children property of *V* has no XQuery element node, then the result of <XML valid predicate> is *False*.

e) If *V* is an XQuery document node and the children property of *V* has at least one XQuery text node, then the result of <XML valid predicate> is *False*.

f) Otherwise:

i) Case:

1) If *V* is an XQuery document node, then let *N* be the number of XQuery element nodes in the children property *CP* of *V*. Let E_i, 1 (one) ≤ *i* ≤ *N*, be an enumeration of these XQuery element nodes.

2) Otherwise, *V* is an XQuery element node. Let *N* be 1 (one), and let *E1* be *V*.

ii) For every *i*, 1 (one) ≤ *i* ≤ *N*:

1) Let D_i be an XQuery document node whose children property consists of only E_i.

2) Case:

A) If <XML valid according to clause> is specified, let SXS_i be the *SXS* chosen by the Syntax Rules, and let $SXSE_i$ be *True*.

B) Otherwise, let NS_i be the namespace of E_i. Let SXS_i be a registered XML Schema for which the current user has USAGE privilege and whose target namespace is identical to NS_i, as defined by [Namespaces], chosen according to a deterministic implementation-defined algorithm that is repeatable, in the sense that if the algorithm is re-evaluated with the same collection of registered XML Schemas that are accessible to the user and the same value *V*, then the same registered XML Schema will be chosen. If SXS_i exists, then let $SXSE_i$ be *True*; otherwise, let $SXSE_i$ be *False*.

3) Case:

A) If $SXSE_i$ is *False*, then let TV_i be *Unknown*.

B) If <XML valid element clause> is specified, and either the [local part] property of E_i is not equivalent to *EN*, or the [namespace name] property of E_i is not identical to *ENSURI*, as defined by [Namespaces], then let TV_i be *False*.

C) Otherwise, the validity of D_i is assessed against SXS_i according to the rules of [Schema1] and [Schema2]. At the beginnning of this validation, let $FOUND_i$ be *True*.

I) When validating an element or attribute *EOA* whose [namespace name] property is identical to some namespace of SXS_i, as defined by [Namespaces], the XML schema components of SXS_i are used to validate *EOA*.

II) When validating an element or attribute *EOA* whose [namespace name] property is not identical to any namespace of SXS_i, as defined by [Namespaces], a registered XML Schema *RXS* is chosen for which the current user has USAGE privilege and whose target namespace is identical to the [namespace name] property of *EOA* as defined by [Namespaces], using a deterministic implementation-defined algorithm that is repeatable, in the sense that if the algorithm is reevaluated with the same collection of registered XML Schemas that are accessible to the user and the same element or attribute, the same registered XML Schema will be chosen.

NOTE 18 — This circumstance can arise if there is a wildcard schema component whose {namespace constraint} property either is "*not*" together with a namespace name, or is "*any*", and whose {process contents} property is "*strict*" or "*lax*". In an XML Schema Document, such a wildcard schema component is represented as an **<xs:any>** or **<xs:anyAttribute>** element

whose namespace attribute is `'##any'`, `'##other'`, or missing, and whose processContent attribute is `'strict'`, `'lax'`, or missing.

1) If the implementation-defined algorithm does not find a registered XML Schema, then let $FOUND_i$ be set to *False*. *EOA* is not regarded as violating any validity constraint of [Schema1] or [Schema2].

2) Otherwise, *RXS* is used to validate *EOA*.

III) Case:

1) If WITH IDENTITY CONSTRAINTS LOCAL is specified, then

Case:

a) If D_i violates some validity constraint of [Schema1] or [Schema2], then let TV_i be *False*.

b) If $FOUND_i$ is *False*, then let TV_i be *Unknown*.

c) Otherwise, let TV_i be *True*.

2) Otherwise,

Case:

a) If D_i violates some validity constraint of [Schema1] or [Schema2], except possibly the identity constraints specified in [Schema1] section 3.11.4, "Identity constraint definition validation rules", and except possibly the validity constraints of [XML] section 3.3.1, "Attribute types", that are named "Validity constraint: ID" and "Validity constraint: IDREF", then let TV_i be *False*.

b) If $FOUND_i$ is *False*, then let TV_i be *Unknown*.

c) Otherwise, let TV_i be *True*.

iii) Case:

1) If WITH IDENTITY CONSTRAINTS GLOBAL is specified, then the identity constraints specified in [Schema1] section 3.11.4, "Identity constraint definition validation rules", are checked for *V*, using the XQuery document node of *V* as the root for purposes of evaluating XPath expressions, and the constraints of [XML] section 3.3.1, "Attribute types", that are named "Validity constraint: ID" and "Validity constraint: IDREF" are checked for *V*, using *V* as the XML document. If any such constraint is violated, then let *TVG* be *False*; otherwise, let *TVG* be *True*.

2) Otherwise, let *TVG* be *True*.

iv) Case:

1) If *TVG* is *False*, or if any TV_i is *False*, then the result of <XML valid predicate> is *False*.

2) If any TV_i is *Unknown*, then the result of <XML valid predicate> is *Unknown*.

3) Otherwise, the result of <XML valid predicate> is *True*.

Conformance Rules

1) Without Feature X141, “IS VALID predicate: data-driven case”, conforming SQL language shall not contain an <XML valid predicate> that does not contain <XML valid according to clause>.

2) Without Feature X142, “IS VALID predicate: ACCORDING TO clause”, conforming SQL language shall not contain an <XML valid predicate> that contains <XML valid according to clause>.

3) Without Feature X143, “IS VALID predicate: ELEMENT clause”, conforming SQL language shall not contain an <XML valid predicate> that contains <XML valid element clause>.

4) Without Feature X144, “IS VALID predicate: schema location”, conforming SQL language shall not contain an <XML valid predicate> that does not contain <XML valid schema location>.

5) Without Feature X145, “IS VALID predicate outside check constraints”, conforming SQL language shall not contain an <XML valid predicate> that is not directly contained in the <search condition> of a <check constraint definition>.

6) Without Feature X151, “IS VALID predicate: without identity constraints”, conforming SQL language shall not contain an <XML valid predicate> that contains or implies WITHOUT IDENTITY CONSTRAINTS.

7) Without Feature X152, “IS VALID predicate: global identity contraints”, conforming SQL language shall not contain an <XML valid predicate> that contains WITHOUT IDENTITY CONSTRAINTS GLOBAL.

8) Without Feature X153, “IS VALID predicate: local identity constraints”, conforming SQL language shall not contain an <XML valid predicate> that contains WITHOUT IDENTITY CONSTRAINTS LOCAL.

9) Without Feature X154, “IS VALID predicate: DOCUMENT clause”, conforming SQL language shall not contain an <XML valid predicate> that that contains an <XML valid identity constraint option> that is DOCUMENT.

(Blank page)

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 19 — 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 X_i be **_x003A_**.

ii) If *EV* is fully escaped, then let X_i be **_x003A_**.

iii) Otherwise, let X_i be T_i.

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

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

Case:

i) If S_1 is the lowercase letter **x**, then let X_1 be **_x0078_**.

ii) If S_1 is the uppercase letter **X**, then let X_1 be **_x0058_**.

e) If either of the following is true:

— The SQL-implementation supports Feature X211, "XML 1.1 support", and either T_i is not a valid XML 1.1 NameChar, or i = 1 (one) and T_1 is not a valid XML 1.1 NameStartChar

— The SQL-implementation does not support Feature X211, "XML 1.1 support", and either T_i is not a valid XML 1.0 NameChar, or i = 1 (one) and T_1 is not a valid XML 1.0 NameStartChar

then:

i) Let U_1, U_2, ..., U_8 be the eight <uppercase hexit>s such that T_i is U+$U_1U_2...U_8$ in the UCS-4 encoding.

ii) Case:

1) If U_1 = **0** (zero), U_2 = **0** (zero), U_3 = **0** (zero), and U_4 = **0** (zero), then let X_i be **_x**$U_5U_6U_7U_8$**_**.

NOTE 20 — This case implies that T_i has a UCS-2 encoding, which is U+$U_5U_6U_7U_8$.

2) Otherwise, let X_i be **_x**$U_3U_4U_5U_6U_7U_8$**_**.

f) Otherwise, let X_i be T_i.

NOTE 21 — That is, any character in *SQLI* that does not occasion a problem as a character in an XML 1.0 NCName or XML 1.1 NCName is simply copied into the result.

5) Let ***XMLN*** be the character string concatenation of X_1, X_2, ..., and 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.

Syntax Rules

None.

Access Rules

None.

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) 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*.

5) 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 22 — "XML visible column" is defined in Subclause 4.10.5, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

6) 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.

7) Case:

   a) If either XSCT or XST contains an XML 1.1 QName that is not an XML 1.0 QName, then let ***VER*** be "**1.1**".

   b) Otherwise, it is implementation-defined whether ***VER*** is "**1.1**" or "**1.0**".

8) The encoding to use for the result(s) of this mapping is implementation-defined. If this encoding is not UTF-8 or UTF-16, then let ***ENCODING*** be the name of the encoding, and let ***ENC*** be

   ```
   encoding = 'ENCODING'
   ```

   Otherwise, let ***ENC*** be the zero-length string.

9) If ***VER*** is not "**1.0**" or *ENC* is not the zero-length string, then let ***XMLDECL*** be

   ```
   <?xml version='VER' ENC ?>
   ```

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

   a) Let ***SQLXMLNS*** be the value of the XML namespace definition provided for the XML namespace prefix **sqlxml** in Table 2, "XML namespace prefixes and their URIs".

   b) Let ***XSDNS*** be the value of the XML namespace definition provided for the XML namespace prefix **xsd** in Table 2, "XML namespace prefixes and their URIs".

   c) 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 />
      ```

   d) 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"
      ```

   e) Case:

i) If ***TABLEFOREST*** is *False* 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 *False* 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 *True* 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 *True* 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>
```

f) Let ***XSR*** be:

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

11) 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.

12) If *DATA* is *True*, 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 2, "XML namespace prefixes and their URIs".

c) Case:

i) If ***TABLEFOREST*** is *False* 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 *False* 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 *True*, then ***XD*** is ***XDROWS***.

d) Case:

i) If ***TABLEFOREST*** is *True*, then let ***XDR*** be:

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

ii) If ***TABLEFOREST*** is *False*, then let ***XDR*** be:

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

13) If *DATA* is *True*, then ***XDR*** is the XML representation of the data of *T*. If *METADATA* is *True*, 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 *True*.

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 *False*.

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 *True*.

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 *True*.

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

   NOTE 23 — "visible column" is defined in Subclause 4.10.5, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

4) Let *CT* be the XML visible columns of the viewed and base tables contained in *S* for *U*.

5) 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.

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 ***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.

8) Case:

   a) If either XSCT or XST contains an XML 1.1 QName that is not an XML 1.0 QName, then let ***VER*** be "**1.1**".

   b) Otherwise, it is implementation-defined whether ***VER*** is "**1.1**" or "**1.0**".

9) The encoding to use for the result(s) of this mapping is implementation-defined. If this encoding is not UTF-8 or UTF-16, then let ***ENCODING*** be the name of the encoding, and let ***ENC*** be

   ```
   encoding = 'ENCODING'
   ```

   Otherwise, let ***ENC*** be the zero-length string.

10) If ***VER*** is not "**1.0**" or *ENC* is not the zero-length string, then let ***XMLDECL*** be

    ```
    <?xml version='VER' ENC ?>
    ```

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

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

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

    c) 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 />
       ```

    d) 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"
       ```

e) 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>
```

f) Let ***XSR*** be:

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

12) 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.

13) 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 2, "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 )
```

14) 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 *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.

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

   NOTE 24 — "visible column" is defined in Subclause 4.10.5, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

4) Let *CT* be the XML visible columns of the viewed and base tables contained in *C* for *U*.

5) 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.

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) 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.

8) Case:

   a) If either XSCT or XST contains an XML 1.1 QName that is not an XML 1.0 QName, then let ***VER*** be "**1.1**".

   b) Otherwise, it is implementation-defined whether ***VER*** is "**1.1**" or "**1.0**".

9) The encoding to use for the result(s) of this mapping is implementation-defined. If this encoding is not UTF-8 or UTF-16, then let ***ENCODING*** be the name of the encoding, and let ***ENC*** be

   ```
   encoding = 'ENCODING'
   ```

   Otherwise, let ***ENC*** be the zero-length string.

10) If ***VER*** is not "**1.0**" or *ENC* is not the zero-length string, then let ***XMLDECL*** be

    ```
    <?xml version='VER' ENC ?>
    ```

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

    a) Let ***SQLXMLNS*** be the value of the XML namespace definition provided for the XML namespace prefix **sqlxml** in Table 2, "XML namespace prefixes and their URIs".

    b) Let ***XSDNS*** be the value of the XML namespace definition provided for the XML namespace prefix **xsd** in Table 2, "XML namespace prefixes and their URIs".

    c) 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 />
       ```

    d) 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"
       ```

    e) 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>
```

f) Let ***XSR*** be:

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

12) 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.

13) If *DATA* is *True*, 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 2, "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 )
```

14) 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 25 — "XML visible column" is defined in Subclause 4.10.5, "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 $\boldsymbol{XMLCE_i}$ 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
      ...
      XMLCEn
   </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 26 — "XML visible table" is defined in Subclause 4.10.5, "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 ***XMLTE$_i$*** be

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


ii) If ***TABLEFOREST*** is *True*, 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 *False*, then let ***XMLSCHEMAT*** be:

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

b) If ***TABLEFOREST*** is *True*, 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 27 — "XML visible schema" is defined in Subclause 4.10.5, "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 ***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 $\boldsymbol{XMLN_i}$ 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 $\boldsymbol{XMLT_i}$ 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 || ***XMLT*$_i$**

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 28 — "XML visible column" is defined in Subclause 4.10.5, "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 2, "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 ***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 ***XMLC*** $_j$ is the zero-length string.

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

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

3) Otherwise:

A) Let ***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), *ENCODING* as the choice of whether to encode binary strings in base64 or in hex, and *True* as *CHARMAPPING*.

B) ***XMLC*** $_j$ is

```
<XMLCN_j>XMLV_j</XMLCN_j>
```

c) Case:

i) If ***TABLEFOREST*** is *False*, then let ***XMLR*** $_i$ be

```
<row>
   XMLC_1
   ...
   XMLC_m
</row>
```

ii) If ***TABLEFOREST*** is *True* and ***TARGETNS*** is the zero-length string, then let ***XMLR*** $_i$ be

```
<XMLTN XSI">
   XMLC_1
   ...
   XMLC_m
</XMLTN>
```

iii) If ***TABLEFOREST*** is *True* and ***TARGETNS*** is not the zero-length string, then let ***XMLR*** $_i$ be

```
<XMLTN XSI xmlns="TARGETNS">
   XMLC_1
   ...
   XMLC_m
</XMLTN>
```

7) Case:

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

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

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

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

   c) If ***TABLEFOREST*** is *True*, 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 29 — "XML visible table " is defined in Subclause 4.10.5, "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 $\boldsymbol{XMLT_i}$ 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 30 — "XML visible schema" is defined in Subclause 4.10.5, "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 $\textbf{\textit{XMLS}}_{\textbf{\textit{i}}}$ 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 2, "XML namespace prefixes and their URIs".

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

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

8) Case:

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

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

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

iii) Let ***NLIT*** and ***MLIT*** be canonical XML Schema literals of the XML Schema type **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 31 — *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

1) Case:

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

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

      II) Let *FMAX* be *FACETMAX*.

      III) Let *FMIN* be *FACETMIN*.

   B) Otherwise:

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

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

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

2) Let *REST* be:


```
<xsd:restriction base="TYPE">
  FMAX
  FMIN
</xsd:restriction>
```

Table 3 — Constraining facets of XML Schema integer types

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

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

```
<xsd:simpleType>
  ANN
  REST
</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 32 — *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 **xdt:yearMonthDuration**.

2) If *SQLT* is a day-time duration, then let ***DTYPE*** be **xdt: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 $\mathbf{XMLMT_i}$ 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 $\mathbf{XMLFN_i}$ 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 $\mathbf{FNLIT_i}$ be the result of mapping FN_i to Unicode using *TM*, for *i* between 1 (one) and *N*.

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

```
<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 other than XML(SEQUENCE), 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 CHARMAPPING be the choice of whether to replace certain characters ("&" (U+0026), "<" (U+003C), ">" (U+003E), and Carriage Return (U+000D)) with their character references (*True*) or not (*False*).

5) Let ***`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.

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

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

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

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

9) Case:

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

   i) Let *CS* be the character set of *SQLT*. Let ***XMLVRAW*** be the 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) Case:

      1) If *CHARMAPPING* is *True*, then let ***XMLV*** be ***XMLVRAW***, with each instance of "&" (U+0026) replaced by "&", each instance of "<" (U+003C) replaced by "<", each instance of ">" (U+003E) replaced by ">", and each instance of Carriage Return (U+000D) replaced by "
".

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

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

   Case:

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

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

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

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

```
LOWER (CV)
```

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

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

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

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

f) If *SQLT* specifies TIME, then:

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

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

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

   iv) 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)
```

10) ***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 X_1, X_2 , ..., 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) ≤ i ≤ N, has not yet been determined, then

   Case:

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

      Case:

      i) If the Unicode code point U+$X_{i+2}X_{i+3}X_{i+4}X_{i+5}$ is a valid Unicode character *UC*, then let U_i be the character string of length 1 (one) whose character is *UC* and let U_{i+1}, U_{i+2}, U_{i+3}), U_{i+4}, U_{i+5}, and U_{i+6} be the zero-length string.

      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 X_{i} = '_' (an <underscore>), and X_{i+1} = '**x**', and each of X_{i+2}, X_{i+3}, X_{i+4}, X_{i+5}, X_{i+6}, and X_{i+7}, are all <hexit>s, and X_{i+8} = '_', then

      Case:

      i) If the Unicode code point U+$X_{i+2}X_{i+3}X_{i+4}X_{i+5}X_{i+6}X_{i+7}$ 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) ≤ *i* ≤ *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 Retrieval assignment

This Subclause modifies Subclause 9.1, "Retrieval assignment", in ISO/IEC 9075-2.

Function

Specify rules for assignments to targets that do not support null values or that support null values with indicator parameters (*e.g.*, assigning SQL-data to host parameters or host variables).

Syntax Rules

1) Replace SR 2) If *TD* is binary string, numeric, boolean, datetime, interval, XML, or a user-defined type, then either *SD* shall be assignable to *TD* or there shall exist an appropriate user-defined cast function *UDCF* from *SD* to *TD*.

Access Rules

No additional Access Rules.

General Rules

1) Augment GR 6) If *TD* is an XML type, then:

   a) Case:

      i) If *TD* is XML(UNTYPED DOCUMENT) or XML(ANY DOCUMENT), and *V* is not a value of type XML(ANY DOCUMENT), then an exception condition is raised: *data exception — invalid XML document*.

      ii) If *TD* is XML(UNTYPED CONTENT) or XML(ANY CONTENT), and *V* is not a value of type XML(ANY CONTENT), then an exception condition is raised: *data exception — invalid XML content*.

   b) Case:

      i) If *TD* is XML(UNTYPED CONTENT) or XML(UNTYPED DOCUMENT) and *SD* is not XML(UNTYPED CONTENT) or XML(UNTYPED DOCUMENT), then:

1) Let *V2* be the result of applying the General Rules of Subclause 10.17, "Constructing a copy of an XML value", with *V* as the *VALUE*.

2) Case:

   A) If the declared type of *T* is XML(UNTYPED CONTENT) or XML(UNTYPED DOCUMENT), then let *V3* be the result of applying the General Rules of Subclause 10.18, "Constructing an unvalidated XQuery document node", to *V2*.

   B) Otherwise, let *V3* be *V2*.

3) *T* is set to *V3*.

ii) Otherwise, *T* is set to *V*.

Conformance Rules

No additional Conformance Rules.

10.2 Store assignment

Function

Specify rules for assignments where the target permits null without the use of indicator parameters or indicator variables, such as storing SQL-data or setting the value of SQL parameters.

Syntax Rules

1) Replace SR 2) If *TD* is binary string, numeric, boolean, datetime, interval, XML, or a user-defined type, then either *SD* shall be assignable to *TD* or there shall exist an appropriate user-defined cast function *UDCF* from *SD* to *TD*.

Access Rules

No additional Access Rules.

General Rules

1) Augment GR 2)b) If the declared type of *T* is an XML type, then:

   a) Case:

      i) If *TD* is XML(UNTYPED DOCUMENT) or XML(ANY DOCUMENT), and *V* is not a value of type XML(ANY DOCUMENT), then an exception condition is raised: *data exception — invalid XML document*.

      ii) If *TD* is XML(UNTYPED CONTENT) or XML(ANY CONTENT), and *V* is not a value of type XML(ANY CONTENT), then an exception condition is raised: *data exception — invalid XML content*.

   b) Case:

      i) If *T* is a <column reference> or a <target array element specification> whose <target array reference> is a <column reference>, or if *TD* is XML(UNTYPED CONTENT) or XML(UNTYPED DOCUMENT) and *SD* is not XML(UNTYPED CONTENT) or XML(UNTYPED DOCUMENT), then:

         1) Let *V2* be the result of applying the General Rules of Subclause 10.17, "Constructing a copy of an XML value", with *V* as the *VALUE*.

         2) Case:

            A) If the declared type of *T* is XML(UNTYPED CONTENT) or XML(UNTYPED DOCUMENT), then let *V3* be the result of applying the General Rules of Subclause 10.18, "Constructing an unvalidated XQuery document node", to *V2*.

            B) Otherwise, let *V3* be *V2*.

         3) *T* is set to *V3*.

      ii) Otherwise, *T* is set to *V*.

Conformance Rules

No additional Conformance Rules.

10.3 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 an XML type, then *TPL* is XML.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.4 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.

10.5 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 before GR 2)d) 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 33 — This definition involves the simultaneous recursive definition of identical XQuery items and identical properties of XQuery nodes, as well as identical XML values.

a) Two XQuery sequences are identical if they have the same number of XQuery items and their corresponding XQuery items are identical.

b) Two XQuery atomic values *XQAV1* and *XQAV2* are identical if they are labeled with the same XQuery atomic type *XQAT* and

Case:

i) If *XQAT* is **`xs:duration`**, or an XML Schema type derived from **`xs:duration`**, then let *YM1* and *YM2* be XQuery atomic values of type **`xdt:yearMonthDuration`** whose year and month components equal the year and month components of *XQAV1* and *XQAV2*, respectively, and let *DT1* and *DT2* be XQuery atomic values of type **`xdt:dayTimeDuration`** whose day, hour, minute, and second components equal the day, hour, minute, and second components of *XQAV1* and *XQAV2*, respectively. *XQAV1* and *XQAV2* are identical values if *YM1* equals *YM2* and *DT1* equals *DT2*.

ii) Otherwise, the result of comparing *XQAV1* and *XQAV2* using the XQuery "eq" operation is *True*.

c) Two XQuery nodes *XII1* and *XII2* are defined to be identical if they satisfy the following conditions:

i) *XII1* and *XII2* are the same kind of XQuery node (both are an XQuery document node, an XQuery element node, an XQuery attribute node, an XQuery processing instruction node, an XQuery text node, an XQuery comment node, or an XQuery namespace node).

ii) If *XIII* is not a document node, then either of the following conditions holds:

1) The parent property of *XII1* and the parent property of *XII2* are both empty.

2) The XQuery node *P1* that is the parent property of *XII1* is identical to the XQuery node *P2* that is the parent property of *XII2*, and if *XII1* and *XII2* are not XQuery attribute nodes or XQuery namespace nodes, then *XII1* and *XII2* have the same ordinal position in the children property of *P1* and *P2*, respectively.

iii) Every significant property *P* of *XII1* is identical to the corresponding property *P* of *XII2*. The significant and insignifcant properties of XQuery nodes are shown in Table 4, "XQuery Node Properties".

Table 4 — XQuery Node Properties

XQuery Node Type	Significant Properties	Insignificant Properties
document	children unparsed entities	base-uri document-uri
element	node-name parent type children attributes nilled	namespaces base-URI
attribute	node-name string value parent type	
processing instruction	target content parent	base-URI
text	content parent	
comment	content parent	
namespace	uri parent	prefix

d) A property *P* of an XQuery node *XII1* is identical to the same property *P* of another XQuery node *XII2* if

Case:

i) If *P* is the children property or the parent property, then the XQuery sequences that are returned by the XQuery accessor of property *P* applied to *XII1* and *XII2* are identical.

ii) If *P* is the unparsed-entities property or the attributes property, then the number of XQuery items returned by the XQuery accessor of property *P* applied to *XII1* equals the number of XQuery items returned by the XQuery accessor of property *P* applied to *XII2*, and there exists a one-to-one mapping of the XQuery items returned by the XQuery accessor of property *P* applied to *XII1* to the XQuery items returned by the XQuery accessor of property *P* applied to *XII1* such that corresponding XQuery items are identical.

iii) If *P* is the node-name property, type property, nilled property, string-value property, uri property, target property, or content property, then the XQuery atomic value that is returned by the XQuery accessor of property *P* applied to *XII1* is identical to the XQuery atomic value that is returned by the XQuery accessor of property *P* applied to *XII2*.

NOTE 34 — The content and uri properties are accessed by the ***dm*:string-value** XQuery accessor, and the target property is accessed by the ***dm*:node-name** XQuery accessor. Otherwise, the name of the XQuery accessor is the same as the name of the property.

Conformance Rules

No additional Conformance Rules.

10.6 Equality operations

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

Function

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

Syntax Rules

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

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.7 Grouping operations

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

Function

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

Syntax Rules

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

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.8 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.

10.9 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.

10.10 Determination of namespace URI

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) Case:

         1) If *P* is equivalent to `'xml'`, then let *NSURI* be "`http://www.w3.org/XML/1998/namespace`".

         2) Otherwise:

            A) *B* shall be within the scope of one or more <XML namespace declaration>s that contain an <XML namespace prefix> equivalent to *P*.

            B) Let *XND* be the <XML namespace declaration> that contains an <XML namespace prefix> equivalent to *P* with innermost scope that includes *B*.

            C) Let *XNDI* be the <XML namespace declaration item> of *XND* that contains an <XML namespace prefix> equivalent to *P*.

            D) 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.

10.11 Construction of an XML element

Function

Construct an XML value consisting of a single XQuery element node.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *QN* be the *QNAME* in an application of this Subclause (an XML 1.1 QName), let *NSURI* be the *NAMESPACE* in an application of this Subclause (an XML namespace URI), let *ATTRS* be the *ATTRIBUTES* in an application of this Subclause (a set of XQuery attribute nodes), let *CON* be the *CONTENTS* in an application of this Subclause (a list of values), let *OPT* be the *OPTION* in an application of this Subclause (an <XML content option>, possibly missing), let *ENC* be the *ENCODING* in an application of this Subclause (an indication of whether binary strings are to be encoded in base64 or hex), and let *XRC* be the *RETURNING* in an application of this Subclause (an <XML returning clause>).

2) Let *N* be the number of values in *CON*. Let V_1, ... , V_N be an enumeration of the values in *CON*.

3) Case:

   a) If $N > 0$ (zero) and, for every i between 1 (one) and N, V_i is null, then let *ALLNULL* be *True*.

   b) Otherwise, let *ALLNULL* be *False*.

4) Case:

   a) If *ALLNULL* is *True* and *OPT* is NULL ON NULL, then the result is the null value.

   b) If *ALLNULL* is *True* and *OPT* is ABSENT ON NULL, then

      Case:

      i) If *XRC* is RETURNING SEQUENCE, then the result is an empty XQuery sequence.

      ii) Otherwise, the result is an XQuery document node with the following properties:

         1) The base-uri property is implementation-defined.

         2) The children property is an empty XQuery sequence.

         3) The unparsed-entities property is the empty set.

         4) The document-uri property is implementation-defined.

   c) Otherwise,

i) For each j, 1 (one) $\leq j \leq N$, such that V_j is not null:

1) Case:

A) If the most specific type of V_j is an XML type, then let CEC_j be a variable whose value is identical to V_j.

B) 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 V_j with *ENC* as the *ENCODING*, "absent" as *NULLS*, and *True* as *CHARMAPPING*. $XMLV_j$ is a character string of Unicode characters. Let CEC_j be the result of

```
XMLPARSE (CONTENT XMLV_j PRESERVE WHITESPACE)
```

2) If *XRC* is RETURNING SEQUENCE, and CEC_j is an XQuery sequence containing an XQuery document node, then a warning condition is raised: *warning — XQuery document node was stripped*.

3) Let S_j be the result of applying Subclause 10.16, "Removing XQuery document nodes from an XQuery sequence", with CEC_j as the SEQUENCE. If *XRC* is RETURNING CONTENT, then it is implementation-defined whether the unparsed-entities property of any XQuery document node removed by this process is copied to the unparsed-entities property of the result.

ii) Let *NILLED* be defined as follows.

Case:

1) If NIL ON NULL is specified, and every V_j is null, then *NILLED* is *True*.

2) If NIL ON NO CONTENT is specified, and there is not at least one V_j that is an XQuery element node or an XQuery text node, then *NILLED* is *True*.

3) Otherwise, *NILLED* is *False*.

iii) Case:

1) If *NILLED* is *True*, then:

A) Let *XSIURI* be the XML namespace URI given in Table 2, "XML namespace prefixes and their URIs", corresponding to the XML namespace prefix **xsi**.

B) Let *NILAIII* be an XQuery attribute node whose properties are set as follows:

I) The node-name property is an XQuery atomic value of XQuery atomic type **xs:QName**, whose local name is "nil" and whose namespace URI is *XSIURI*.

II) The string-value property is "true".

III) The type property is **xdt:untypedAtomic**.

IV) The parent property is *EI*.

2) Otherwise, let *NILAII* be the empty XQuery sequence.

iv) Let *S* be the XQuery sequence formed by concatenating *ATTRS*, *NILAII*, and the S_j in order from $j = 1$ (one) through $j = N$.

v) Let *XSC* be an XQuery static context created by applying the Syntax Rules of Subclause 10.19, "Creation of an XQuery expression context", with the <XML element> or <XML forest> that invoked the current Subclause as the *BNF*. The validation mode of *XSC* is set to an implementation-defined value.

vi) Let *XDC* be a XQuery dynamic context created by applying the General Rules of Subclause 10.19, "Creation of an XQuery expression context".

vii) Let *XSC* and *XDC* be augmented with the following XQuery variables:

1) An XQuery variable whose name is *QNAME*, whose XQuery formal type notation is **`xs:QName`**, and whose value is a QName whose local name is the XML 1.1 QName local part of *EN* and whose namespace URI is *ENSURI*.

2) An XQuery variable whose name is *SEQ*, whose XQuery formal type notation is **`( element of type xs:anyType | attribute of type xs:anySimpleType | text | comment | processing-instruction | xdt:anyAtomicType | document { ( element of type xs:anyType | comment | processing-instruction | text ) * } ) *`** and whose value is *S*.

viii) Let *EI* be the result of an XQuery evaluation with XML 1.1 lexical rules, using *XSC* and *XDC* as the XQuery expression context, of the XQuery expression

```
element { $QNAME } { $SEQ }
```

If an XQuery error occurs during the evaluation, then an exception condition is raised: *XQuery error*.

NOTE 35 — If the implementation does not support Feature X211, "XML 1.1 support", then any Names, NCNames, and QNames found in either $QNAME or $SEQ will already be XML 1.0 Names, NCNames, and QNames, and the result of the XQuery evaluation does not depend on which XML lexical rules are followed.

ix) The result is

Case:

1) If *XRC* is RETURNING SEQUENCE, then *EI*.

2) Otherwise, an XQuery document node with the following properties:

A) The base-uri property is implementation-defined.

B) The children property consists of *EI*.

C) The unparsed-entities property is the empty set.

D) The document-uri property is implementation-defined.

Conformance Rules

None.

10.12 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 36 — As a set, duplicates are removed from *SQ*.

4) Let *N* be the number of elements of *SQ*. Let M_i, 1 (one) $\le i \le N$, 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 name] property of *EII*.

ii) Otherwise,

Case:

1) If M_i is equivalent to P, then the [namespace name] property of *EII*.

2) Otherwise, the [namespace name] 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) ≤ $i \le N$.

Conformance Rules

None.

10.13 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, and let *XRC* be the *RETURNING* 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 *XRC* is RETURNING SEQUENCE, then

      Case:

      i) If *X* is the null value, then *C* is *Y*.

      ii) If *Y* is the null value, then *C* is *X*.

      iii) Otherwise, *C* is the XQuery sequence consisting of every XQuery item of *X*, in order, followed by every XQuery item of *Y*, in order.

   b) Otherwise,

      Case:

      i) If *X* is the null value and *Y* is the null value, then *C* is the null value.

      ii) Otherwise:

         1) If *X* is null, then let *X1* be the empty XQuery sequence; otherwise, let *X1* be the result of applying the General Rules of Subclause 10.16, "Removing XQuery document nodes from an XQuery sequence", with *X* as the *SEQUENCE*.

         2) If *Y* is null, then let *Y1* be the empty XQuery sequence; otherwise, let *Y1* be the result of applying the General Rules of Subclause 10.16, "Removing XQuery document nodes from an XQuery sequence", with *Y* as the *SEQUENCE*.

         3) If any XQuery node of *X1* or *Y1* is an XQuery attribute node or an XQuery namespace node, then an exception condition is raised: *data exception — invalid XML document*.

         4) Let *C* be an XQuery document node with the following properties:

A) The base-URI and document-uri properties are implementation-defined.

B) The children property consists of the XQuery nodes of *X1*, in order, followed by the XQuery nodes of *Y1*, in order.

C) The unparsed-entities property is the empty set.

3) *C* is the result of the concatenation of *X* and *Y*.

Conformance Rules

None.

10.14 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, CONTENT, or SEQUENCE). Let *DT* be the string type *TYPE* in an application of this Subclause. Let *CS* be the *ENCODING* in an application of this Subclause. Let *VP* be the *VERSION* in an application of this Subclause.

2) Case:

   a) If *V* is the null value, then the result is the null value.

   b) Otherwise,

      Case:

      i) If *DT* is a character string type, then

         Case:

         1) If *DC* is SEQUENCE, then:

            A) Let *CV* be the result of applying Section 4, "XML output method", of [Serialization] to *V*. If this produces a serialization error, or if the result is otherwise undefined, then an exception condition is raised: *data exception — XQuery sequence serialization error*.

            B) The serialization of *V* is the result of the assignment of *CV* to a target of type *DT* according to the rules of .

         2) Otherwise,

            A) If *V* is not an XQuery sequence of length 1 (one) whose sole XQuery item is an XQuery document node, then an exception condition is raised: *data exception — not an XQuery document node*.

            B) Let *D* be the XQuery document node that is the sole XQuery item of *V*.

            C) Case:

               I) If *DC* is DOCUMENT, then:

1) If the children property of *D* does not contain exactly one XQuery element node, then an exception condition is raised: *data exception — not an XML document*.

2) Case:

   a) If *CS* is UTF8 or UTF16, then let *CEC* be the zero-length string.

   b) If *CS* is UTF32, then let *CEC* be

      encoding="UTF-32"

   c) Otherwise, let *EN* be the implementation-defined name of the character encoding of *CS*. Let *CEC* be

      encoding="*EN*"

3) It is implementation-defined whether *SA* is

   standalone="yes"

   or

   standalone="no"

   or the zero-length string.

4) Case:

   a) If *VP* is not "1.0", then let *VA* be

      version="*VP*"

   b) If either *CEC* or *SA* is not the zero-length string, then let *VA* be

      version="1.0"

   c) Otherwise, let *VA* be the zero-length string.

5) If *VA* is not the zero-length string, then let *CV1* be

   CAST ('<?xml *VA CEC SA* ?>' AS *DT*)

II) Otherwise, let *CV1* be the zero-length string.

D) 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*.

NOTE 37 — [Canonical XML] is an example of an algorithm that may be used to compute *CVTEMP*.

NOTE 38 — Whether certain characters, (such as "<") are handled by means of entity references (such as <) or by use of CDATA sections, is implementation-dependent.

E) 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*.

F) The serialization of *V* is the result of the assignment of

```
CV1 || CV2
```

to a target of type *DT* according to the rules of Subclause 9.2, "Store assignment", in ISO/IEC 9075-2.

ii) Otherwise (*DT* is a binary string type),

Case:

1) If *DC* is SEQUENCE, then:

A) Let *CV* be a binary string equivalent to the result of applying Section 4, "XML output method", of [Serialization] to *V*. If this produces a serialization error, or if the result is otherwise undefined, then an exception condition is raised: *data exception — XQuery sequence serialization error*.

B) The serialization of *V* is the result of the assignment of *CV* to a target of type *DT* according to the rules of .

2) Otherwise,

A) If *V* is not an XQuery sequence of length 1 (one) whose sole XQuery item is an XQuery document node, then an exception condition is raised: *data exception — not an XQuery document node*.

B) Let *D* be the XQuery document node that is the sole XQuery item of *V*.

C) Case:

I) If *DC* is DOCUMENT, then:

1) If the children property of *D* does not contain exactly one XQuery element node, then an exception condition is raised: *data exception — not an XML document*.

2) Case:

a) If *CS* is UTF8 or UTF16, then let *CEC* be the zero-length string.

b) If *CS* is UTF32, then let *CEC* be

```
encoding="UTF-32"
```

c) Otherwise, let *EN* be the implementation-defined name of the character encoding of *CS*. Let *CEC* be

```
encoding="EN"
```

3) It is implementation-defined whether *SA* is

standalone="yes"

or

standalone="no"

or the zero-length string.

4) Case:

a) If *VP* is not "1.0", then let *VA* be

version="*VP*"

b) If either *CEC* or *SA* is not the zero-length string, then let *VA* be

version="1.0"

c) Otherwise, let *VA* be the zero-length string.

5) If *VA* is not the zero-length string, then let *CV1* be a binary string equivalent to

<?xml *VA CEC SA* ?>

II) Otherwise, let *CV1* be the zero-length string.

D) Case:

I) If *CS* is UTF16, then let *BOM* be a binary string equivalent to `U&'\FEFF'`.

II) Otherwise, it is implementation-defined whether *BOM* is a binary string equivalent to `U&'\FEFF'` or the zero-length string.

NOTE 39 — The purpose of the previous rule is to ensure that a Byte Order Mark (BOM) can be prefixed to the serialized value when needed because of the specified encoding.

E) Let *CV2* be an implementation-dependent binary string such that the result of

XMLPARSE (*DC* (*BOM* || *CV1* || *CV2*) PRESERVE WHITESPACE)

is identical to *V*. If there is no such value *CV2*, then an exception condition is raised: *data exception — character not in repertoire*.

F) The serialization of *V* is the result of the assignment of

BOM* || *CV1* || *CV2

to a target of type *DT* according to the rules of Subclause 9.2, "Store assignment", in ISO/IEC 9075-2.

Conformance Rules

None.

10.15 Parsing a string as an XML value

Function

Parse a character string or a binary 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 string *TEXT*, and let *WO* be the value of the *OPTION* (either PRESERVE WHITESPACE or STRIP WHITESPACE) in an application of this Subclause.

2) A string is called a *textual XML 1.0 document* if the string conforms to the definition of a well-formed XML document as defined in [XML 1.0], as modified by [Namespaces 1.0].

3) A string is called a *textual XML 1.1 document* if the string conforms to the definition of a well-formed XML document as defined in [XML 1.1], as modified by [Namespaces 1.1].

4) A string is called a *textual XML 1.0 content* if the following are all true:

   a) The string matches the following production as an extension to the grammar of [XML 1.0], as modified by [Namespaces 1.0]:

      ```
      XMLvalue ::= XMLDecl? content
      ```

   b) The string meets all the well-formedness constraints of applicable productions of [XML 1.0], as modified by [Namespaces 1.0].

   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].

5) A string is called a *textual XML 1.1 content* if the following are all true:

   a) The string matches the following production as an extension to the grammar of [XML 1.1], as modified by [Namespaces 1.1]:

      ```
      XMLvalue ::= XMLDecl? content
      ```

   b) The string meets all the well-formedness constraints of applicable productions of [XML 1.1], as modified by [Namespaces 1.1].

   c) Each of the parsed entities, as defined by [XML 1.1], that is referenced directly or indirectly within the textual XML content is well-formed, as defined by [XML 1.1].

6) Case:

   a) If the SQL-implementation supports Feature X211, "XML 1.1 support", then

      Case:

      i) If *DC* is DOCUMENT and *V* is neither a textual XML 1.0 document nor a textual XML 1.1 document, then an exception condition is raised: *data exception — invalid XML document*.

      ii) If *DC* is CONTENT and *V* is neither a textual XML 1.0 content nor a textual XML 1.1 content, then an exception condition is raised: *data exception — invalid XML content*.

   b) Otherwise,

      Case:

      i) If *DC* is DOCUMENT and *V* is not a textual XML 1.0 document, then an exception condition is raised: *data exception — invalid XML document*.

      ii) If *DC* is CONTENT and *V* is not a textual XML 1.0 content, then an exception condition is raised: *data exception — invalid XML content*.

7) *V* is parsed and a collection *C* of XML information items is produced according to the rules of [Infoset], with the following modifications:

   a) The generation of the XML document information item *XRII* is modified as follows:

      i) If *DC* is CONTENT, then the [children] property of the XML document information item 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] or of [XML 1.1].

      NOTE 40 — It is not an error for the XML document information item to contain zero or more than one XML element information item in this case, or for it to contain XML character information items.

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

      iii) The [unparsed entities] property is implementation-defined.

   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.

8) If *WO* is STRIP WHITESPACE, then:

   a) An XML element information item *EII* contained in *C* is *potentially whitespace-strippable* if any of the following is true:

      i) *EII* is contained in the [children] property of *XRII* and *EII* does not have an [attributes] property that contains an XML attribute information item for which all of the following are true:

         1) The [local name] property is "space".

         2) The [namespace name] property is `http://www.w3.org/XML/1998/namespace`.

         3) The [normalized value] property is "preserve".

ii) *EII* has an [attributes] property that contains an XML attribute information item for which all of the following are true:

1) The [local name] property is “space”.

2) The [namespace name] property is `http://www.w3.org/XML/1998/namespace`.

3) The [normalized value] property is “default”

iii) *EII* is contained in the [children] property of a potentially whitespace-strippable XML element information item and *EII* does not have an [attributes] property that contains an XML attribute information item for which all of the following is true:

1) The [local name] property is “space”.

2) The [namespace name] property is `http://www.w3.org/XML/1998/ namespace`.

3) The [normalized value] property is “preserve”.

b) For every potentially whitespace-strippable XML element information item *PWSEII* contained in *C*:

i) Let *N* be the cardinality of the [children] property of *PWSEII*. Let XII_i, 1 (one) ≤ *i* ≤ *N*, be the list of XML information items that is the [children] property of *PWSEII*.

ii) For every *i* between 1 (one) and *N*, if XII_i is an XML character information item, then:

1) Let *j* be the least subscript less than or equal to *i* such that for all *q* between *j* and *i*, XII_q is an XML character information item.

2) Let *k* be the greatest subscript greater than or equal to *i* such that for all *q* between *i* and *k*, XII_q is an XML character information item.

NOTE 41 — Thus the list XII_j, ... , XII_k is the maximal sublist of XII_1, ... , XII_N 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”.

3) 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 XII_q is marked for removal from *C*.

iii) For every *i* between 1 (one) and *N*, if XII_i is marked for removal from *C*, then XII_i is removed from *C*.

c) Let *N* be the cardinality of the [children] property of *XRII*. Let XII_i, 1 (one) ≤ *i* ≤ *N*, be the list of XML information items that is the [children] property of *XRII*.

i) For every *i* between 1 (one) and *N*, if XII_i is an XML character information item, then:

1) Let *j* be the least subscript less than or equal to *i* such that for all *q* between *j* and *i*, XII_q is an XML character information item.

2) Let *k* be the greatest subscript greater than or equal to *i* such that for all *q* between *i* and *k*, XII_q is an XML character information item.

3) 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 XII_q is marked for removal from *C*.

ii) For every *i* between 1 (one) and *N*, if XII_i is marked for removal from *C*, then XII_i is removed from *C*.

9) *C* is converted to an XQuery document node *D* (and its children) according to the rules of [XQuery DM] section 6, "Nodes".

10) *D* is the result of parsing *V* as an XML value.

Conformance Rules

None.

10.16 Removing XQuery document nodes from an XQuery sequence

Function

Replace each XQuery document node in a sequence by the sequence of children of the XQuery document node.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *S* be the *SEQUENCE* in an application of this Subclause.
2) Case:
   a) If *S* is the null value, then the result of this Subclause is the null value.
   b) Otherwise:
      i) Let *N* be the number of XQuery items in *S*.
      ii) Let I_j, 1 (one) ≤ j ≤ *N*, be an enumeration in order of the XQuery items in *S*.
      iii) For each j, 1 (one) ≤ j ≤ *N*,

           Case:

           1) If I_j is an XQuery document node, then let C_j be the XQuery sequence that is the children property of I_j.
           2) Otherwise, let C_j be I_j.

      iv) Let *R* be the concatenation of the XQuery sequences C_j in order from j = 1 (one) through j = *N*.
      v) *R* is the result of this Subclause.

Conformance Rules

None.

10.17 Constructing a copy of an XML value

Function

Construct a copy of an XML value.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *V* be the *VALUE* (a value of an XML type) in an application of this Subclause.

2) Let *N* be the number of XQuery items in *V*.

3) Let I_j, 1 (one) ≤ j ≤ *N*, be an enumeration in order of the XQuery items in *V*.

4) For each j, 1 (one) ≤ j ≤ *N*, let C_j be defined as follows:

   Case:

   a) If I_j is an XQuery atomic value, then let C_j be I_j.

   b) Otherwise, let C_j be a value of XML type that is identical to I_j except that the parent property (if any) of C_j is empty.

5) Let *R* be the XQuery sequence enumerated by C_j, 1 (one) ≤ j ≤ *N*.

6) *R* is the result of this Subclause.

Conformance Rules

None.

10.18 Constructing an unvalidated XQuery document node

Function

Convert an XQuery document node to a value of type XML(UNTYPED CONTENT).

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *XDN* be the XQuery document node in an application of this Subclause.

2) Let *R* be an XQuery document node that is identical to *XDN* except:

   a) The type property of every XQuery element node in the XQuery tree rooted by *R* is "**`xdt:untypedAny`**".

   b) The nilled property of every XQuery element node in the XQuery tree rooted by *R* is "false".

   c) The type property of every XQuery attribute node in the XQuery tree rooted by *R* is "**`xdt:untypedAtomic`**".

3) *R* is the result of this Subclause.

Conformance Rules

None.

10.19 Creation of an XQuery expression context

Function

Create an XQuery expression context.

Syntax Rules

1) Let *BNF* be the BNF non-terminal (possibly omitted) specified in an invocation of this Subclause.

2) Let *XSC* be an XQuery static context, created as follows:

   a) *XSC* is initially created as defined by [XQuery] Appendix C.1, "Static context components", as specified in the table titled "Static context components" in the column headed "Default predefined value".

   b) *XSC* is modified with implementation-defined values, as permitted by [XQuery] Appendix C.1, "static context components", in the column headed "Can be overwritten or augmented by implementation?".

   c) If *BNF* is specified, then:

      i) The in-scope namespaces component of *XSC* is overwritten or augmented with certain namespaces, as follows.

         1) For each <XML namespace prefix> *XNP* contained in an <XML namespace declaration> whose scope contains *BNF*, let *XND* be the <XML namespace declaration> with innermost scope containing *BNF*, and let *XNURI* be the <XML namespace URI> of the <XML regular namespace declaration item> containing *XNP* in *XND*.

         2) The pair (*XNP*, *XNURI*) is placed in the in-scope namespaces component of *XSC*, either:

            A) If *XNP* is previously defined, then overwriting the previously defined namespace.

            B) Otherwise, augmenting the in-scope namespaces component of *XSC*.

      NOTE 42 — The scope of <XML namespace declaration>s is defined in the various Subclauses that reference that nonterminal symbol, such as Subclause 6.11, "<XML element>", and Subclause 7.2, "<query expression>".

      ii) If there is an <XML namespace declaration> whose scope includes *BNF* and that contains an <XML default namespace declaration item>, then let *XND* be the innermost such <XML namespace declaration>.

      Case:

      1) If *XND* contains an <XML default namespace declaration item> of the form "DEFAULT <XML namespace URI>", then let *XNURI* be that <XML namespace URI>.

         Case:

         A) If *XNURI* is the zero-length string, then the default element/type namespace component of *XSC* is set to empty.

         B) Otherwise, the default element/type namespace component of *XSC* is set to *XNURI*.

      2) If *XND* contains an <XML default namespace declaration item> of the the form "NO DEFAULT", then the default element/type namespace component of *XSC* is set to empty.

d) The in-scope schema definitions component of *XSC* (*i.e.*, the in-scope type definitions, the in-scope element declarations, and the in-scope attribute declarations) consists of an implementation-defined collection of registered XML schemas for which the current user has USAGE privilege.

Access Rules

None.

General Rules

1) Let *XDC* be an XQuery dynamic context whose components are initially set as specified in [XQuery] Appendix C.2, "Dynamic context components", in the table titled "Dynamic context components", in the columns titled "Default predefined value" and "Can be overwritten or augmented by implementation?".

2) *XDC* is the result of this Subclause.

Conformance Rules

None.

10.20 Determination of an XQuery formal type notation

Function

Determine the XQuery formal type notation of an XQuery variable in the XQuery static context.

Syntax Rules

1) Let *S* be the *SOURCE* in an invocation of this Subclause.

2) Let *SD* be the declared type of *S*.

3) Case:

   a) If *S* is a column reference that references a known not null column, then let ***BOUNDED*** be the empty string.

   b) Otherwise, let ***BOUNDED*** be "**?**".

4) Let *XFTN* be the XQuery formal type notation determined as follows.

   Case:

   a) If *SD* is XML(UNTYPED DOCUMENT), then let ***XFTN*** be "**document { element of type xdt:untypedAny} *BOUNDED***".

   b) If *SD* is XML(ANY DOCUMENT), then let ***XFTN*** be "**document { element of type xdt:anyType} *BOUNDED***".

   c) If *SD* is XML(UNTYPED CONTENT), then let ***XFTN*** be "**document { (element of type xdt:untypedAny | comment | processing-instruction | text) * } *BOUNDED***".

   d) If *SD* is XML(ANY CONTENT), then let ***XFTN*** be "**document { (element of type xdt:anyType | comment | processing-instruction | text) * } *BOUNDED***".

   e) If *SD* is XML(SEQUENCE), then let ***XFTN*** be "**document { (element of type xs:anyType | attribute of type xs:anySimpleType | text | comment | processing-instruction | xdt:anyAtomicType | document { (element of type xs:anyType | | comment | processing-instruction | text) * }) ***".

   f) If *SD* is an SQL predefined type, then let *XMLT* be the result of applying the General Rules of Subclause 9.15, "Mapping SQL data types to XML Schema data types", with *SD* as the SQL type, *ENC* as the *ENCODING*, and "absent" as the *NULLS*. Let *PT* be the XML Schema primitive type from which *XMLT* is derived.

      Case:

      i) If *SD* is a year-month interval, then let ***XFTN*** be "**xdt:yearMonthDuration *BOUNDED***".

      ii) If *SD* is a day-time interval, then let ***XFTN*** be "**xdt:dayTimeDuration *BOUNDED***".

      iii) If *SD* is an exact numeric type with scale 0 (zero), then let ***XFTN*** be "**xs:integer *BOUNDED***".

      iv) Otherwise, let ***XFTN*** be "***PT BOUNDED***".

5) ***XFTN***, or an implementation-defined XML Schema subtype of ***XFTN*** that describes *S*, is the result of this Subclause.

NOTE 43 — For example, an implementation may use “empty” as the XQuery formal type notation for XMLCAST (EMPTY AS XML(SEQUENCE)).

Access Rules

None.

General Rules

None.

Conformance Rules

None.

(Blank page)

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

1) Insert after SR 8)c)i)6) If P_i is an SQL parameter of declared type XML(UNTYPED CONTENT) whose <XML passing mechanism> is BY REF, then the declared type of A_i shall be either XML(UNTYPED CONTENT) or XML(UNTYPED DOCUMENT).

2) Insert after SR 8)c)i)6) If P_i is an SQL parameter of declared type XML(UNTYPED DOCUMENT) whose <XML passing mechanism> is BY REF, then the declared type of A_i shall be either XML(UNTYPED DOCUMENT).

3) Insert after SR 8)c)ii)6) If P_i is an SQL parameter of declared type XML(UNTYPED CONTENT) whose <XML passing mechanism> is BY REF, then the declared type of A_i shall be either XML(UNTYPED CONTENT) or XML(UNTYPED DOCUMENT).

4) Insert after SR 8)c)ii)6) If P_i is an SQL parameter of declared type XML(UNTYPED DOCUMENT) whose <XML passing mechanism> is BY REF, then the declared type of A_i shall be either XML(UNTYPED DOCUMENT).

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 3)a) If P_i is an input SQL parameter or both an input SQL parameter and an output SQL parameter, then

   Case:

   a) If the <XML passing mechanism> of P_i is BY VALUE, then let CPV_i be the result of an application of Subclause 10.17, "Constructing a copy of an XML value", with V_i as the *VALUE*.

   b) Otherwise, let CPV_i be the result of the assignment of V_i to a target of type T_i according to the rules of Subclause 10.2, "Store assignment".

2) Insert after GR 4)b)i)1) If T_i is an 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

      ```
      BOMi || XMLSERIALIZE (XOi CPVi AS XDTi)
      ```

3) Insert after GR 4)b)ii)1) If T_i is an 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.

4) Insert before GR 8)h)i)4)B)II)2)b) If *CRT* is an 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)
      ```

5) Insert before GR 8)i)iii)2)C) If T_i is an 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)
      ```

6) Replace GR 9)a)iii) Let the result of the <routine invocation> be

   Case:

a) If the <XML passing mechanism> of the <returns clause> of *R* is BY VALUE, then the result of an application of Subclause 10.17, "Constructing a copy of an XML value", with *RV* as the *VALUE*.

b) Otherwise, the result of assigning *RV* to a target of declared type *ERDT* according to the rules of Subclause 10.2, "Store assignment".

7) Replace GR 9)b)ii)1)B)V)1)b) The *I*-th element of *A* is set to

Case:

a) If the <XML passing mechanism> of P_i is BY VALUE, then the result of an application of Subclause 10.17, "Constructing a copy of an XML value", with V_i as the *VALUE*.

b) Otherwise, the value of CPV_i , denoted by *SV*, by applying the General Rules of Subclause 10.2, "Store assignment", to the *I*-th element of *A* and *SV* as *TARGET* and *VALUE*, respectively.

8) Replace GR 9)b)ii)1)B)V)2)c) The *I*-th element of *A* is set to

Case:

a) If the <XML passing mechanism> of P_i is BY VALUE, then the result of an application of Subclause 10.17, "Constructing a copy of an XML value", with V_i as the *VALUE*.

b) Otherwise, the value of CPV_i, denoted by *SV*, by applying the General Rules of Subclause 10.2, "Store assignment", to the *I*-th element of *A* and *SV* as *TARGET* and *VALUE*, respectively.

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> ]
    [ <XML returning clause> ]
    <right paren>
```

Syntax Rules

1) Insert this SR If <XML returning clause> is not specified, then RETURNING CONTENT is implicit.

2) Insert this SR The declared type of <XML aggregate> is:

   a) If RETURNING CONTENT is specified or implied, then

      Case:

      i) If the declared type of the <XML value expression> is XML(UNTYPED DOCUMENT) or XML(UNTYPED CONTENT), then XML(UNTYPED CONTENT).

      ii) Otherwise, XML(ANY CONTENT).

   b) Otherwise, XML(SEQUENCE).

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 an XML type, 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 *XRC* be the explicit or implicit <XML returning clause>.

      3) Let V_i, 2 ≤ i ≤ N, be the result of applying the General Rules of Subclause 10.13, "Concatenation of two XML values", with V_i and the value of the first column of R_i as the XML values and *XRC* as the *RETURNING*.

      4) The result is *VN*.

Conformance Rules

1) Insert this CR Without Feature X034, "XMLAgg", conforming SQL language shall not contain an <XML aggregate>.

2) 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>.

3) Insert this CR Without Feature X241, "Explicit RETURNING CONTENT in XML publishing", in Conforming SQL language, an <XML aggregate> shall not specify an <XML returning clause> that is RETURNING CONTENT.

4) Insert this CR Without Feature X242, "Explicit RETURNING SEQUENCE in XML publishing", in Conforming SQL language, an <XML aggregate> shall not specify an <XML returning clause> that is RETURNING SEQUENCE.

11.3 <XML lexically scoped options>

Function

Declare one or more XML namespaces and the encoding to use for binary strings.

Format

```
<XML lexically scoped options> ::=
    <XML lexically scoped option> [ <comma> <XML lexically scoped option> ]

<XML lexically scoped 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 regular namespace declaration item>
  | <XML default namespace declaration item>

<XML namespace prefix> ::= <identifier>

<XML namespace URI> ::= <character string literal>

<XML regular namespace declaration item> ::= <XML namespace URI> AS <XML namespace prefix>

<XML default namespace declaration item> ::=
    DEFAULT <XML namespace URI>
  | NO DEFAULT

<XML binary encoding> ::=
    XMLBINARY [ USING ] { BASE64 | HEX }
```

Syntax Rules

1) An <XML lexically scoped 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 1.1 NCName.

4) No two <XML namespace prefix>es shall be equivalent.

5) No <XML namespace prefix> shall be equivalent to **`xml`** or **`xmlns`**.

6) No <XML namespace URI> shall be identical, as defined in [Namespaces], to "`http://www.w3.org/2000/xmlns/`" or to "`http://www.w3.org/XML/1998/namespace`".

7) The value of an <XML namespace URI> contained in an <XML regular namespace declaration item> shall not be a zero-length string.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature X211, "XML 1.1 support", in conforming SQL language, each <XML namespace prefix> shall be an XML 1.0 NCName.

11.4 <XML returning clause>

Function

Specify whether the declared type of certain <XML value function>s is XML(SEQUENCE) or some other XML type.

Format

```
<XML returning clause> ::= RETURNING { CONTENT | SEQUENCE }
```

Syntax Rules

None.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

11.5 <XML passing mechanism>

Function

Specify the semantics for assigning an XML value when passing an XML value to or from XMLQuery or an SQL routine.

Format

```
<XML passing mechanism> ::=

    BY REF
  | BY VALUE
```

Syntax Rules

None.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature X221, “XML passing mechanism BY VALUE”, conforming SQL language shall not contain an <XML passing mechanism> that is BY VALUE.

2) Without Feature X222, “XML passing mechanism BY REF”, conforming SQL language shall not contain an <XML passing mechanism> that is BY REF.

(Blank page)

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 lexically scoped options> ]
    <value expression> <right paren>
```

Syntax Rules

1) Insert this SR The scope of each <XML namespace declaration item> contained in the <XML lexically scoped options> is the <generation expression>.

2) Insert this SR The scope of an <XML binary encoding> contained in the <XML lexically scoped 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 lexically scoped options> that contains an <XML namespace declaration>.

2) Insert this CR Without Feature X133, “XMLBINARY clause in DDL”, in conforming SQL language, a <generation expression> shall not immediately contain an <XML lexically scoped options> that contains an <XML binary encoding>.

3) 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 an XML type or a distinct type whose source type is an XML type.

4) Insert this CR Without Feature X251, “Persistent XML values of XML(UNTYPED DOCUMENT) type”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(UNTYPED DOCUMENT) type or a distinct type whose source type is the XML(UNTYPED DOCUMENT) type.

5) Insert this CR Without Feature X252, “Persistent XML values of XML(ANY DOCUMENT) type”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(ANY DOCUMENT) type or a distinct type whose source type is the XML(ANY DOCUMENT) type.

6) Insert this CR Without Feature X253, “Persistent XML values of XML(UNTYPED CONTENT) type”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(UNTYPED CONTENT) type or a distinct type whose source type is the XML(UNTYPED CONTENT) type.

7) Insert this CR Without Feature X254, “Persistent XML values of XML(ANY CONTENT) type”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(ANY CONTENT) type or a distinct type whose source type is the XML(ANY CONTENT) type.

8) Insert this CR Without Feature X255, “Persistent XML values of XML(SEQUENCE) type”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(SEQUENCE) type or a distinct type whose source type is the XML(SEQUENCE) 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 lexically scoped options> ]
    <search condition> <right paren>
```

Syntax Rules

1) Insert this SR The scope of each <XML namespace declaration item> contained in the <XML lexically scoped option> is the <check constraint definition>.

2) Insert this SR The scope of an <XML binary encoding> contained in the <XML lexically scoped options> is the <check constraint 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, a <check constraint definition> shall not immediately contain and <XML lexically scoped options> that contains an <XML namespace declaration>.

2) Insert this CR Without Feature X133, "XMLBINARY clause in DDL", in conforming SQL language, a <check constraint definition> shall not immediately contain an <XML lexically scoped 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 an XML type or a distinct type whose source type is an XML type.

2) Insert this CR Without Feature X251, "Persistent XML values of XML(UNTYPED DOCUMENT) type", conforming SQL language shall not contain a <view definition> whose declared type is based on either the XML(UNTYPED DOCUMENT) type or a distinct type whose source type is the XML(UNTYPED DOCUMENT) type.

3) Insert this CR Without Feature X252, "Persistent XML values of XML(ANY DOCUMENT) type", conforming SQL language shall not contain a <view definition> whose declared type is based on either the XML(ANY DOCUMENT) type or a distinct type whose source type is the XML(ANY DOCUMENT) type.

4) Insert this CR Without Feature X253, "Persistent XML values of XML(UNTYPED CONTENT) type", conforming SQL language shall not contain a <view definition> whose declared type is based on either the XML(UNTYPED CONTENT) type or a distinct type whose source type is the XML(UNTYPED CONTENT) type.

5) Insert this CR Without Feature X254, “Persistent XML values of XML(ANY CONTENT) type”, conforming SQL language shall not contain a <view definition> whose declared type is based on either the XML(ANY CONTENT) type or a distinct type whose source type is the XML(ANY CONTENT) type.

6) Insert this CR Without Feature X255, “Persistent XML values of XML(SEQUENCE) type”, conforming SQL language shall not contain a <view definition> whose declared type is based on either the XML(SEQUENCE) type or a distinct type whose source type is the XML(SEQUENCE) 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 lexically scoped options> ] <search condition> <right paren>
```

Syntax Rules

1) Insert this SR The scope of each <XML namespace declaration item> contained in the <XML lexically scoped options> is the <assertion definition>.

2) Insert this SR The scope of an <XML binary encoding> contained in the <XML lexically scoped 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 lexically scoped options> that contains an <XML namespace declaration>.

2) Insert this CR Without Feature X133, "XMLBINARY clause in DDL", in conforming SQL language, an <assertion definition> shall not immediately contain an <XML lexically scoped 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) 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 an XML type.

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 an XML type or a distinct type whose source type is an 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

```
<SQL parameter declaration> ::=
    [ <parameter mode> ]
    [ <SQL parameter name> ]
    <parameter type>
    [ RESULT ]
    [ <XML passing mechanism> ]

<parameter type> ::=
    <data type> [ <locator indication> ]
    [ <document or content> ] [ <string type option> ]

<returns clause> ::=
    RETURNS <returns type> [ <XML passing mechanism> ]

<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 an XML type 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 an 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 an 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 an 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 $RFT_{i\text{-}PN}$ is an XML type, then:

a) Let *XDT* be the data type identified by the <character string type> contained in <string type option> immediately contained in $RFT_{i\text{-}PN}$. *XDT* is the associated string type of the result of *R*.

b) Let *XO* be the <document or content> immediately contained in $RFT_{i\text{-}PN}$. *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 an 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 an 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*.

8) Insert this SR Case:

a) If *R* is an external routine, then <SQL parameter declaration> and <returns clause> shall not contain an <XML passing mechanism>.

b) Otherwise:

i) If the declared type of an <SQL parameter declaration> is an XML type or a distinct type whose source type is an XML type, and the <SQL parameter declaration> does not contain an <XML passing mechanism>, then BY VALUE is implicit.

ii) If the declared type of an <SQL parameter declaration> is not an XML type or a distinct type whose source type is an XML type, then the <SQL parameter declaration> shall not contain an <XML passing mechanism>.

iii) If *R* is an SQL-invoked function, the declared type of the <returns type> is an XML type or a distinct type whose source type is an XML type, and the <returns clause> does not contain an <XML passing mechanism>, then BY VALUE is implicit.

iv) If *R* is an SQL-invoked function and the declared type of the <returns type> is not an XML type or a distinct type whose source type is an XML type, then the <returns clause> shall not contain an <XML passing mechanism>.

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.

3) Insert after GR 3)q) For every SQL parameter whose <SQL parameter declaration> contains an explicit or implicit <XML passing mechanism>, an indication of that <XML passing mechanism>.

4) Insert after GR 3)r) If *R* is an SQL function, then an indication of the explicit or implicit <XML passing mechanism> contained in the <returns clause>.

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 an XML type or a distinct type whose source type is an 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 an XML type or a distinct type whose source type is an 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.

12.8 <user-defined cast definition>

This Subclause modifes Subclause 11.53, "<user-defined cast definition>", of ISO/IEC 9075-2.

Function

Define a user-defined cast.

Format

```
No additional Format items.
```

Syntax Rules

1) Neither *SDT* nor *SDT*TDT shall be a distinct type whose source type is an XML type.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

(Blank page)

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";
DATA_EXCEPTION_NOT_AN_XQUERY_DOCUMENT_NODE:
  constant SQLSTATE_TYPE := "2200U";
DATA_EXCEPTION_INVALID_XQUERY_CONTEXT_ITEM:
  constant SQLSTATE_TYPE := "2200V";
DATA_EXCEPTION_XQUERY_SEQUENCE_SERIALIZATION_ERROR:
  constant SQLSTATE_TYPE := "2200W";
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";
XQUERY_ERROR_NO_SUBCLASS:
  constant SQLSTATE_TYPE := "10000";
```

```
WARNING_COLUMN_CANNOT_BE_MAPPED:
  constant SQLSTATE_TYPE := "01010";
WARNING_XQUERY_DOCUMENT_NODE_WAS_STRIPPED:
  constant SQLSTATE_TYPE := "01011";
```

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

Augment Table 16, "Data type correspondences for Ada"

Table 5 — Data type correspondences for Ada

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

Augment Table 17, "Data type correspondences for C"

Table 6 — Data type correspondences for C

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

Augment Table 18, "Data type correspondences for COBOL"

Table 7 — Data type correspondences for COBOL

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

Augment Table 19, "Data type correspondences for Fortran"

Table 8 — Data type correspondences for Fortran

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

Augment Table 20, "Data type correspondences for M"

Table 9 — Data type correspondences for M

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

Augment Table 21, "Data type correspondences for Pascal"

Table 10 — Data type correspondences for Pascal

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

Augment Table 22, "Data type correspondences for PL/I"

Table 11 — 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.

(Blank page)

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 lexically scoped 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 lexically scoped options> is the <delete statement: searched>.

2) Insert this SR The scope of an <XML binary encoding> contained in the <XML lexically scoped 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 lexically scoped options> that contains an <XML namespace declaration>.

2) Insert this CR Without Feature X132, “XMLBINARY clause in DML”, in conforming SQL language, a <delete statement: searched> shall not immediately contain an <XML lexically scoped 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 lexically scoped 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 lexically scoped options> is the <insert statement>.

2) Insert this SR The scope of an <XML binary encoding> contained in <XML lexically scoped 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 lexically scoped options> that contains an <XML namespace declaration>.

2) Insert this CR Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, an <insert statement> shall not immediately contain an <XML lexically scoped 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 lexically scoped options> ] INTO <target table>
    [ [ AS ] <correlation name> ]
    USING <table reference> ON <search condition>
    <merge operation specification>
```

Syntax Rules

1) Insert this SR The scope of each <XML namespace declaration item> contained in <XML lexically scoped options> is the <merge statement>.

2) Insert this SR The scope of an <XML binary encoding> contained in <XML lexically scoped options> is the <merge 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, a <merge statement> shall not immediately contain an <XML lexically scoped options> that contains an <XML namespace declaration>.

2) Insert this CR Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, a <merge statement> shall not immediately contain an <XML lexically scoped 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 lexically scoped 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 lexically scoped options> is the <update statement: positioned>.

2) Insert this SR The scope of an <XML binary encoding> contained in <XML lexically scoped 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 lexically scoped options> that contains an <XML namespace declaration>.

2) Insert this CR Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, an <update statement: positioned> shall not immediately contain an <XML lexically scoped 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 lexically scoped 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 lexically scoped options> is the <update statement: searched>.

2) Insert this SR The scope of an <XML binary encoding> contained in <XML lexically scoped 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 lexically scoped options> that contains an <XML namespace declaration>.

2) Insert this CR Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, an <update statement: searched> shall not immediately contain an <XML lexically scoped options> that contains an <XML binary encoding>.

15 Control statements

This Clause modifies Clause 15, "Control statements", in ISO/IEC 9075-2.

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 lexically scoped 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 lexically scoped options> is the <compound statement>.

2) Insert this SR The scope of an <XML binary encoding> contained in <XML lexically scoped options> is the <compound statement>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) 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 lexically scoped options> that contains an <XML namespace declaration>.

2) Insert this CR Without Feature X134, "XMLBINARY clause in compound statements", in conforming SQL language, a <compound statement> shall not immediately contain an <XML lexically scoped options> that contains an <XML binary encoding>.

15.2 <return statement>

This Subclause modifies Subclause 15.2, "<return statement>", in ISO/IEC 9075-2.

Function

Return a value from an SQL function.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert after SR 3) If *RDT* is XML(UNTYPED CONTENT) and the <XML passing mechanism> contained in the <returns clause> of *F* is BY REF, then the declared type of *VE* shall be either XML(UNTYPED CONTENT) or XML(UNTYPED DOCUMENT).

2) Insert after SR 3) If *RDT* is XML(UNTYPED DOCUMENT) and the <XML passing mechanism> contained in the <returns clause> of *F* is BY REF, then the declared type of *VE* shall be XML(UNTYPED DOCUMENT).

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

(Blank page)

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.

(Blank page)

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 12, "Codes used for SQL data types in Dynamic SQL", specifies the codes associated with the SQL data types.

Augment Table 25, "Codes used for SQL data types in Dynamic SQL"

Table 12 — 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 an 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 6.8, "<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 6.8, "<XML value function>", then an exception condition is raised in accordance with the General Rules of Subclause 6.8, "<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 6.6, "<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 6.6, "<string value function>", then an exception condition is raised in accordance with the General Rules of Subclause 6.6, "<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) ≤ i ≤ 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) ≤ i ≤ 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) ≤ i ≤ n, let XPI_j, 1 (one) ≤ j ≤ d, be the input host parameters, let XPO_k, 1 (one) ≤ k ≤ e, be the output host parameters, and let $XPIO_l$, 1 (one) ≤ l f, be the host parameters that are both input host parameters and output host parameters.

   c) For 1 (one) ≤ j ≤ 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) ≤ k ≤ 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) ≤ l ≤ 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 $IOBOM_l$ 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 XVIOf = XMLPARSE (SDOCIOf XPNIOf STRIP WHITESPACE);
    NES;
    SET PNO1 = CAST ( FSON1 (SVO1) AS TSO1);
    ...
    SET PNOb = CAST ( FSONb (SVOb) AS TSOb);
```



```
    SET XPNO1 = OBOM1 || XMLSERIALIZE ( XDOCO1 XVO1 AS XTO1 );
    ...
    SET XPNOe = OBOMe || XMLSERIALIZE ( XDOCOe XVOe AS XTOe );
    SET PNIO1 = CAST ( FSION1 (SVIO1) AS TSIO1);
    ...
    SET PNIOc = CAST ( FSIONc (SVIOc) AS TSIOc);
    SET XPNIO1 = IOBOM1 || XMLSERIALIZE ( XDOCIO1 XVIO1 AS XTIO1 );
    ...
    SET XPNIOf = IOBOMf || XMLSERIALIZE ( XDOCIOf XVIOf AS XTIOf );
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 9075-2.

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 6.6, "<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) 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) 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) 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 9075-2.

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 6.6, "<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 6.6, "<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 9075-2.

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 6.6, "<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) 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) 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) 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 9075-2.

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 6.6, "<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 9075-2.

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 6.6, "<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) 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) 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) 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 9075-2.

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 6.6, "<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) 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) 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) 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 9075-2.

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 6.6, "<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 6.6, "<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) 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.

(Blank page)

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 13 — 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 NCNAME domain

Function

Define a domain that contains an NCNAME.

Definition

```
CREATE DOMAIN NCNAME AS
  CHARACTER VARYING(L)
    CHARACTER SET CS;

GRANT USAGE ON NCNAME
    TO PUBLIC WITH GRANT OPTION;
```

Description

1) It is implementation-defined whether this domain specifies all variable-length character string values that conform to the rules for formation and representation of an XML 1.0 NCName or an XML 1.1 NCName.

   NOTE 44 — There is no way in SQL to specify a <domain constraint> that would be true for an XML 1.0 NCName or an XML 1.1 NCName and false for all other character string values.

2) *L* is the implementation-defined maximum length of an XML 1.0 NCName or XML 1.1 NCName.

3) *CS* is an implementation-defined character set whose character repertoire is UCS.

Conformance Rules

1) Without Feature F251, "Domain support", conforming SQL language shall not reference INFORMATION_SCHEMA.NCNAME.

20.2 URI domain

Function

Define a domain that contains a URI.

Definition

```
CREATE DOMAIN URI AS
  CHARACTER VARYING(L)
    CHARACTER SET CS;

GRANT USAGE ON URI
    TO PUBLIC WITH GRANT OPTION;
```

Description

1) This domain specifies all variable-length character string values that conform to the rules for formation and representation of an <XML URI>.

2) *L* is the implementation-defined maximum length of an <XML URI>.

3) *CS* is an implementation-defined character set whose character repertoire is UCS.

Conformance Rules

1) Without Feature F251, “Domain support”, conforming SQL language shall not reference INFORMATION_SCHEMA.URI.

20.3 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, P1.XML_PASSING_MECHANISM
  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.

20.4 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, R.XML_RETURN_MECHANISM
    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 45 — 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.

20.5 XML_SCHEMA_ELEMENTS view

Function

Identify the global element declaration schema components defined in registered XML Schemas that are accessible to a given user or role.

Definition

```
CREATE VIEW XML_SCHEMA_ELEMENTS AS
  SELECT XS.XML_SCHEMA_TARGET_NAMESPACE, XS.XML_SCHEMA_LOCATION,
         XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME,
         XE.XML_SCHEMA_NAMESPACE, XE.XML_SCHEMA_ELEMENT,
         XE.XML_SCHEMA_ELEMENT_IS_DETERMINISTIC
  FROM DEFINITION_SCHEMA.XML_SCHEMA_ELEMENTS AS XE
  JOIN DEFINITION_SCHEMA.XML_SCHEMAS AS XS
  USING ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME )
  WHERE ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA,
          XML_SCHEMA_NAME, 'XML SCHEMA' )
       IN ( SELECT UP.OBJECT_CATALOG, UP.OBJECT_SCHEMA,
                   UP.OBJECT_NAME, UP.OBJECT_TYPE
            FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES AS UP
            WHERE ( UP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                 OR
                    UP.GRANTEE IN ( SELECT ROLE_NAME
                                    FROM ENABLED_ROLES ) ) )
    AND XML_SCHEMA_CATALOG =
        ( SELECT CATALOG_NAME
          FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE XML_SCHEMAS
    TO PUBLIC PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature X160, “Information Schema for registered XML Schemas”, conforming SQL languages shall not reference INFORMATION_SCHEMA.XML_SCHEMA_ELEMENTS.

2) Without Feature F391, “Long identifiers”, conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCHEMA_ELEMENTS.

20.6 XML_SCHEMA_NAMESPACES view

Function

Identify the namespaces that are defined in registered XML Schemas that are accessible to a given user or role.

Definition

```
CREATE VIEW XML_SCHEMA_NAMESPACES AS
    SELECT XS.XML_SCHEMA_TARGET_NAMESPACE, XS.XML_SCHEMA_LOCATION,
           XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME,
           XSN.XML_SCHEMA_NAMESPACE
    FROM DEFINITION_SCHEMA.XML_SCHEMA_NAMESPACES AS XSN
    JOIN DEFINITION_SCHEMA.XML_SCHEMAS AS XS
    USING ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME )
    WHERE ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA,
            XML_SCHEMA_NAME, 'XML SCHEMA' )
        IN ( SELECT UP.OBJECT_CATALOG, UP.OBJECT_SCHEMA,
                    UP.OBJECT_NAME, UP.OBJECT_TYPE
             FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES AS UP
             WHERE ( UP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                  OR
                     UP.GRANTEE IN ( SELECT ROLE_NAME
                                     FROM ENABLED_ROLES ) ) )
      AND XML_SCHEMA_CATALOG =
          ( SELECT CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE XML_SCHEMAS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature X160, “Information Schema for registered XML Schemas”, conforming SQL languages shall not reference INFORMATION_SCHEMA.XML_SCHEMA_NAMESPACES.

2) Without Feature F391, “Long identifiers”, conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCHEMA_NAMESPACES.

20.7 XML_SCHEMAS view

Function

Identify the registered XML Schemas that are accessible to a given user or role.

Definition

```
CREATE VIEW XML_SCHEMAS AS
    SELECT XML_SCHEMA_TARGET_NAMESPACE, XML_SCHEMA_LOCATION,
    XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME,
    XML_SCHEMA_IS_DETERMINISTIC, XML_SCHEMA_IS_PERMANENT
FROM DEFINITION_SCHEMA.XML_SCHEMAS
WHERE ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA,
        XML_SCHEMA_NAME, 'XML SCHEMA' )
     IN ( SELECT UP.OBJECT_CATALOG, UP.OBJECT_SCHEMA,
                 UP.OBJECT_NAME, UP.OBJECT_TYPE
          FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES AS UP
          WHERE ( UP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
       OR
          UP.GRANTEE IN ( SELECT ROLE_NAME
                          FROM ENABLED_ROLES ) ) )
   AND XML_SCHEMA_CATALOG =
       ( SELECT CATALOG_NAME
         FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE XML_SCHEMAS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature X160, “Information Schema for registered XML Schemas”, conforming SQL languages shall not reference INFORMATION_SCHEMA.XML_SCHEMAS.

2) Without Feature F391, “Long identifiers”, conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCHEMAS.

20.8 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
           XML_PASSING_MECH ) 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,
           XML_PASSING_MECHANISM
    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
           XML_RETURNS_MECH ) 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,
          XML_RETURN_MECHANISM
    FROM INFORMATION_SCHEMA.ROUTINES;

GRANT SELECT ON TABLE ROUTINES_S
    TO PUBLIC WITH GRANT OPTION;
```

Insert the following view definitions:

```
CREATE VIEW XML_SCH_ELEMENTS_S
         ( XML_SCH_TGT_NSPACE, XML_SCH_LOCATION,   XML_SCHEMA_CATALOG,
           XML_SCHEMA_SCHEMA,  XML_SCHEMA_NAME,    XML_SCH_NSPACE,
           XML_SCHEMA_ELEMENT, XML_SCH_EL_IS_DET ) AS
    SELECT XML_SCHEMA_TARGET_NAMESPACE, XML_SCHEMA_LOCATION, XML_SCHEMA_CATALOG,
           XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE,
           XML_SCHEMA_ELEMENT, XML_SCHEMA_ELEMENT_IS_DETERMINISTIC
    FROM INFORMATION_SCHEMA.XML_SCHEMA_ELEMENTS;

GRANT SELECT ON TABLE XML_SCH_ELEMENTS_S
    TO PUBLIC WITH GRANT OPTION;


CREATE VIEW XML_SCH_NSPACES_S
         ( XML_SCH_TGT_NSPACE, XML_SCH_LOCATION,   XML_SCHEMA_CATALOG,
           XML_SCHEMA_SCHEMA,  XML_SCHEMA_NAME,    XML_SCH_NSPACE ) AS
    SELECT XML_SCHEMA_TARGET_NAMESPACE, XML_SCHEMA_LOCATION, XML_SCHEMA_CATALOG,
           XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE
    FROM INFORMATION_SCHEMA.XML_SCHEMA_NAMESPACES;

GRANT SELECT ON TABLE XML_SCH_NSPACES_S
    TO PUBLIC WITH GRANT OPTION;


CREATE VIEW XML_SCHEMAS_S
         ( XML_SCH_TGT_NSPACE, XML_SCH_LOCATION,   XML_SCHEMA_CATALOG,
           XML_SCHEMA_SCHEMA,  XML_SCHEMA_NAME,    XML_SCHEMA_IS_DET,
           XML_SCHEMA_IS_PERM ) AS
    SELECT XML_SCHEMA_TARGET_NAMESPACE, XML_SCHEMA_LOCATION, XML_SCHEMA_CATALOG,
           XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_IS_DETERMINISTIC,
           XML_SCHEMA_IS_PERMANENT
    FROM INFORMATION_SCHEMA.XML_SCHEMAS;

GRANT SELECT ON TABLE XML_SCHEMAS_S
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Insert this CR Without Feature X160, “Information Schema for registered XML Schemas”, conforming SQL languages shall not reference INFORMATION_SCHEMA.XML_SCH_ELEMENTS_S.

2) Insert this CR Without Feature X160, “Information Schema for registered XML Schemas”, conforming SQL languages shall not reference INFORMATION_SCHEMA.XML_SCH_NSPACES_S.

3) Insert this CR Without Feature X160, “Information Schema for registered XML Schemas”, conforming SQL languages shall not reference INFORMATION_SCHEMA.XML_SCHEMAS_S.

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 IS IN
    ( 'XML(UNTYPED CONTENT)',
      'XML(UNTYPED DOCUMENT)',
      'XML(ANY CONTENT)',
      'XML(ANY DOCUMENT)',
      'XML(SEQUENCE)' ) )
  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'
      'XML(UNTYPED CONTENT)',
      'XML(UNTYPED DOCUMENT)',
      'XML(ANY CONTENT)',
      'XML(ANY DOCUMENT)',
      'XML(SEQUENCE)' ) ) ),
```

Description

1) Insert this Description If DATA_TYPE is 'XML(UNTYPED CONTENT)', then the data type being described is XML(UNTYPED CONTENT).

2) Insert this Description If DATA_TYPE is 'XML(UNTYPED DOCUMENT)', then the data type being described is XML(UNTYPED DOCUMENT).

3) Insert this Description If DATA_TYPE is 'XML(ANY CONTENT)', then the data type being described is XML(ANY CONTENT).

4) Insert this Description If DATA_TYPE is 'XML(ANY DOCUMENT)', then the data type being described is XML(ANY DOCUMENT).

5) Insert this Description If DATA_TYPE is 'XML(SEQUENCE)', then the data type being described is XML(SEQUENCE).

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' ) ),
XML_PASSING_MECHANISM      INFORMATION_SCHEMA.CHARACTER_DATA
  CONSTRAINT XML_PASSING_MECHANISM_CHECK
    CHECK (XML_PASSING_MECHANISM IN
      ( 'BY REF', 'BY VALUE' ) ),
```

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.

3) Insert this Description The values of XML_PASSING_MECHANISM have the following meanings:

null	The parameter has no <XML passing mechanism>.
BY REF	The <XML passing mechanism> of the parameter is BY REF.
BY VALUE	The <XML passing mechanism> of the parameter is BY VALUE.

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' ) ),
XML_RETURN_MECHANISM       INFORMATION_SCHEMA.CHARACTER_DATA
  CONSTRAINT XML_RETURN_MECHANISM_CHECK
    CHECK (XML_RETURN_MECHANISM IN
      ( 'BY REF', 'BY VALUE' ) ),
```

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.

3) Insert this Description The values of XML_RETURN_MECHANISM have the following meanings:

null	The SQL-invoked routine does not have a <returns clause>, or its <returns clause> does not have an <XML passing mechanism>.
BY REF	The <XML passing mechanism> of the <returns clause> is BY REF.
BY VALUE	The <XML passing mechanism> of the <returns clause> is BY VALUE.

21.4 USAGE_PRIVILEGES base table

This Subclause modifies Subclause 6.60, "USAGE_PRIVILEGES base table", in ISO/IEC 9075-11.

Function

The USAGE_PRIVILEGES table has one row for each usage privilege descriptor. It effectively contains a representation of the usage privilege descriptors.

Definition

Replace constraint USAGE_PRIVILEGES_OBJECT_TYPE_CHECK

```
      CONSTRAINT USAGE_PRIVILEGES_OBJECT_TYPE_CHECK
        CHECK ( OBJECT_TYPE IN
              ( 'DOMAIN', 'CHARACTER SET', 'COLLATION',
                'TRANSLATION', 'SEQUENCE', 'XML SCHEMA' ) ),
```

Replace constraint USAGE_PRIVILEGES_CHECK_REFERENCES_OBJECT

```
    CONSTRAINT USAGE_PRIVILEGES_CHECK_REFERENCES_OBJECT
      CHECK ( ( OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE ) IN
              ( SELECT DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME, 'DOMAIN'
                FROM DOMAINS
              UNION
                SELECT CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
                       'CHARACTER SET'
                FROM CHARACTER_SETS
              UNION
                SELECT COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME,
                       'COLLATION'
                FROM COLLATIONS
              UNION
                SELECT TRANSLATION_CATALOG, TRANSLATION_SCHEMA, TRANSLATION_NAME,
                       'TRANSLATION'
                FROM TRANSLATIONS
              UNION
                SELECT SEQUENCE_CATALOG, SEQUENCE_SCHEMA, SEQUENCE_NAME,
                       'SEQUENCE'
                FROM SEQUENCES
              UNION
                SELECT XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME,
                       'XML SCHEMA'
                FROM XML_SCHEMAS ) ),
```

Description

1) Augment Descr. 3).

XML SCHEMA	The object to which the privilege applies is a registered XML Schema.

21.5 XML_SCHEMA_ELEMENTS base table

Function

The XML_SCHEMA_ELEMENTS base table has one row for each global element declaration component of each registered XML Schema.

Definition

```
CREATE TABLE XML_SCHEMA_ELEMENTS (
    XML_SCHEMA_CATALOG                      INFORMATION_SCHEMA.SQL_IDENTIFIER,
    XML_SCHEMA_SCHEMA                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
    XML_SCHEMA_NAME                         INFORMATION_SCHEMA.SQL_IDENTIFIER,
    XML_SCHEMA_NAMESPACE                    INFORMATION_SCHEMA.URI,
    XML_SCHEMA_ELEMENT                      INFORMATION_SCHEMA.NCNAME,
    XML_SCHEMA_ELEMENT_IS_DETERMINISTIC     INFORMATION_SCHEMA.YES_OR_NO
      CONSTRAINT XML_SCHEMA_ELEMENT_IS_DETERMINISTIC_NOT_NULL
        NOT NULL,

    CONSTRAINT XML_SCHEMA_ELEMENTS_PRIMARY_KEY
      PRIMARY KEY ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA,
                    XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE,
                    XML_SCHEMA_ELEMENT ),

    CONSTRAINT XML_SCHEMA_ELEMENTS_FOREIGN_KEY_XML_SCHEMA_NAMESPACES
      FOREIGN KEY ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA,
                    XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE )
      REFERENCES XML_SCHEMA_NAMESPACES )
```

Description

1) The values of XML_SCHEMA_CATALOG and XML_SCHEMA_SCHEMA are the catalog name and the unqualified schema name, respectively, of the registered XML Schema containing the global element declaration schema component being described.

2) The value of XML_SCHEMA_NAME is the name of the registered XML Schema containing the global element declaration schema component being described.

3) The value of XML_SCHEMA_NAMESPACE is the namespace URI of the global element declaration schema component being described.

4) The value of XML_SCHEMA_ELEMENT is the XML QName local part of the name of the global element declaration schema component being described.

5) The values of XML_SCHEMA_IS_DETERMINISTIC have the following meanings:

YES	The global element declaration schema component being described is not non-deterministic.

NO	The global element declaration schema component being described is non-deterministic.

21.6 XML_SCHEMA_NAMESPACES base table

Function

The XML_SCHEMA_NAMESPACES base table has one row for each namespace of each registered XML Schema descriptor.

Definition

```
CREATE TABLE XML_SCHEMA_NAMESPACES (
    XML_SCHEMA_CATALOG                  INFORMATION_SCHEMA.SQL_IDENTIFIER,
    XML_SCHEMA_SCHEMA                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
    XML_SCHEMA_NAME                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
    XML_SCHEMA_NAMESPACE                INFORMATION_SCHEMA.URI,

    CONSTRAINT XML_SCHEMA_NAMESPACES_PRIMARY_KEY
      PRIMARY KEY ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA,
                    XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE ),

    CONSTRAINT XML_SCHEMA_NAMESPACES_FOREIGN_KEY_XML_SCHEMAS
      FOREIGN KEY ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA,
                    XML_SCHEMA_NAME )
        REFERENCES XML_SCHEMAS )
```

Description

1) The values of XML_SCHEMA_CATALOG and XML_SCHEMA_SCHEMA are the catalog name and the unqualified schema name, respectively, of the registered XML Schema containing the namespace being described.

2) The value of XML_SCHEMA_NAME is the name of the registered XML Schema containing the namespace being described.

3) The value of XML_SCHEMA_NAMESPACE is the namespace URI of the namespace being described.

21.7 XML_SCHEMAS base table

Function

The XML_SCHEMAS base table has one row for each registered XML Schema.

Definition

```
CREATE TABLE XML_SCHEMAS (
    XML_SCHEMA_TARGET_NAMESPACE      INFORMATION_SCHEMA.URI
      CONSTRAINT XML_SCHEMA_TARGET_NAMESPACE_NOT_NULL
        NOT NULL,
    XML_SCHEMA_LOCATION              INFORMATION_SCHEMA.URI
      CONSTRAINT XML_SCHEMA_LOCATION_NOT_NULL
        NOT NULL,
    XML_SCHEMA_CATALOG               INFORMATION_SCHEMA.SQL_IDENTIFIER,
    XML_SCHEMA_SCHEMA                INFORMATION_SCHEMA.SQL_IDENTIFIER,
    XML_SCHEMA_NAME                  INFORMATION_SCHEMA.SQL_IDENTIFIER,
    XML_SCHEMA_IS_DETERMINISTIC      INFORMATION_SCHEMA.YES_OR_NO
      CONSTRAINT XML_SCHEMA_IS_DETERMINISTIC_NOT_NULL
        NOT NULL,
    XML_SCHEMA_IS_PERMANENT          INFORMATION_SCHEMA.YES_OR_NO
      CONSTRAINT XML_SCHEMA_IS_PERMANENT_NOT_NULL
        NOT NULL,

    CONSTRAINT XML_SCHEMAS_PRIMARY_KEY
      PRIMARY KEY ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME ),

    CONSTRAINT XML_SCHEMAS_ALTERNATE_KEY
      UNIQUE ( XML_SCHEMA_TARGET_NAMESPACE, XML_SCHEMA_LOCATION ),

    CONSTRAINT XML_SCHEMA_FOREIGN_KEY_XML_SCHEMA_NAMESPACES
      FOREIGN KEY ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA,
                    XML_SCHEMA_NAME, XML_SCHEMA_TARGET_NAMESPACE )
        REFERENCES XML_SCHEMA_NAMESPACES,

    CONSTRAINT XML_SCHEMA_FOREIGN_KEY_XML_SCHEMATA
      FOREIGN KEY ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA )
        REFERENCES SCHEMATA )
```

Description

1) The value of XML_SCHEMA_TARGET_NAMESPACE is the target namespace URI of the registered XML Schema being described.

2) The value of XML_SCHEMA_LOCATION is the schema location URI of the registered XML Schema being described.

3) The values of XML_SCHEMA_CATALOG and XML_SCHEMA_SCHEMA are the catalog name and the unqualified schema name, respectively, of the registered XML Schema being described.

4) The value of XML_SCHEMA_NAME is the name of the registered XML Schema being described.

5) The values of XML_SCHEMA_IS_DETERMINISTIC have the following meanings:

YES	The registered XML Schema being described is not non-deterministic.
NO	The registered XML Schema being described is non-deterministic.

6) The values of XML_SCHEMA_IS_PERMANENT have the following meanings:

YES	The registered XML Schema being described is permanently registered.
NO	The registered XML Schema being described is not permanently registered.

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.

(Blank page)

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.

Augment Table 32, "SQLSTATE class and subclass values"

Table 14 — 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
			not an XQuery document node	00U
			invalid XQuery context item	00V
			XQuery sequence serialization error	00W
X	*SQL/XML mapping error*	0N	(*no subclass*)	000
			unmappable XML Name	001

Category	Condition	Class	Subcondition	Subclass
			invalid XML character	002
X	*XQuery error*	10	(*no subclass*)	000
W	*warning*	01	(*no subclass*)	000
			column cannot be mapped	010
			XQuery document node was stripped	011

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: Character string serialization and CONTENT option".

   b) Feature X071, "XMLSerialize: Character string serialization and 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 15 — 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
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
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

Feature ID	Feature Name	Implied Feature ID	Implied Feature Name
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: Character string input and CONTENT option	X010	XML type
X061	XMLParse: Character string input and DOCUMENT option	X010	XML type
X065	XMLParse: BLOB input and CONTENT option	X010	XML type
X066	XMLParse: BLOB input and DOCUMENT option	X010	XML type
X070	XMLSerialize: Character string serialization and CONTENT option	X010	XML type
X071	XMLSerialize: Character string serialization and DOCUMENT option	X010	XML type
X072	XMLSerialize: Character string serialization and SEQUENCE option	X010	XML type
X073	XMLSerialize: BLOB serialization and CONTENT option	X010	XML type
X074	XMLSerialize: BLOB serialization and DOCUMENT option	X010	XML type

Feature ID	Feature Name	Implied Feature ID	Implied Feature Name
X075	XMLSerialize: BLOB serialization and SEQUENCE option	X010	XML type
X076	XMLSerialize: VERSION 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
X091	XML content predicate	X190	XML(SEQUENCE) type
X095	XMLTable	X010	XML type
X096	XMLExists	X010	XML type
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

Feature ID	Feature Name	Implied Feature ID	Implied Feature Name
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
X141	IS VALID predicate: data-driven case	X145	IS VALID predicate outside check constraints
X142	IS VALID predicate: ACCORDING TO clause	X010	XML type
X143	IS VALID predicate: ELEMENT clause	X142	IS VALID predicate: ACCORDING TO clause
X144	IS VALID predicate: schema location	X142	IS VALID predicate: ACCORDING TO clause
X145	IS VALID predicate outside check constraints	X010	XML type
X151	IS VALID predicate: without identity constraints	X010	XML type
X152	IS VALID predicate: global identity constraints	X010	XML type
X153	IS VALID predicate: local identity constraints	X010	XML type
X154	IS VALID predicate: DOCUMENT clause	X010	XML type
X160	Information Schema for registered XML Schemas	X010	XML type
X171	NIL ON NO CONTENT option	X170	XML null handling options
X181	XML(UNTYPED DOCUMENT) type	X010	XML type
X182	XML(ANY DOCUMENT) type	X010	XML type
X190	XML(SEQUENCE) type	X010	XML type
X201	XMLQuery: RETURNING CONTENT	X010	XML type

Feature ID	Feature Name	Implied Feature ID	Implied Feature Name
X202	XMLQuery: RETURNING SEQUENCE	X190	XML(SEQUENCE) type
X211	XML 1.1 support	X010	XML type
X221	XML passing mechanism BY VALUE	X010	XML type
X222	XML passing mechanism BY REF	X010	XML type
X231	XML(UNTYPED CONTENT) type	X010	XML type
X232	XML(ANY CONTENT) type	X010	XML type
X241	Explicit RETURNING CONTENT in XML publishing	X010	XML type
X242	RETURNING SEQUENCE in XML publishing	X190	XML(SEQUENCE) type
X251	Persistent XML values of XML(UNTYPED DOCUMENT) type	X181	XML(UNTYPED DOCUMENT) type
X252	Persistent XML values of XML(ANY DOCUMENT) type	X182	XML(ANY DOCUMENT) type
X253	Persistent XML values of XML(UNTYPED CONTENT) type	X231	XML(UNTYPED CONTENT) type
X254	Persistent XML values of XML(ANY CONTENT) type	X232	XML(ANY CONTENT) type
X255	Persistent XML values of XML(SEQUENCE) type	X190	XML(SEQUENCE) type

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 F251, "Domain support":
   a) Subclause 20.1, "NCNAME domain":
      i) Without Feature F251, "Domain support", conforming SQL language shall not reference INFORMATION_SCHEMA.NCNAME.
   b) Subclause 20.2, "URI domain":
      i) Without Feature F251, "Domain support", conforming SQL language shall not reference INFORMATION_SCHEMA.URI.
2) Specifications for Feature F391, "Long identifiers":
   a) Subclause 20.5, "XML_SCHEMA_ELEMENTS view":
      i) Without Feature F391, "Long identifiers", conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCHEMA_ELEMENTS.
   b) Subclause 20.6, "XML_SCHEMA_NAMESPACES view":
      i) Without Feature F391, "Long identifiers", conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCHEMA_NAMESPACES.
   c) Subclause 20.7, "XML_SCHEMAS view":
      i) Without Feature F391, "Long identifiers", conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCHEMAS.
3) 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>.
4) 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 6.7, "<XML value expression>":

i) Without Feature X010, "XML type", conforming SQL language shall not contain an <XML value expression>.

c) Subclause 6.8, "<XML value function>":

i) Without Feature X010, "XML type", conforming SQL language shall not contain an <XML value function>.

5) 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 an XML type or a distinct type whose source type is an XML type.

6) 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 an XML type or a distinct type whose source type is an XML type.

7) 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 an XML type.

8) 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 an XML type or a distinct type whose source type is an XML type.

9) 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 an XML type or a distinct type whose source type is an XML type.

10) 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 an XML type or a distinct type whose source type is an 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 an XML type or a distinct type whose source type is an XML type.

11) Specifications for Feature X020, “XML concatenation”:

a) Subclause 6.10, “<XML concatenation>”:

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

12) Specifications for Feature X031, “XMLElement”:

a) Subclause 6.11, “<XML element>”:

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

13) Specifications for Feature X032, “XMLForest”:

a) Subclause 6.12, “<XML forest>”:

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

14) 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>.

15) 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>.

16) Specifications for Feature X036, “XMLComment”:

a) Subclause 6.9, “<XML comment>”:

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

17) Specifications for Feature X037, “XMLPI”:

a) Subclause 6.14, “<XML PI>”:

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

18) 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.

19) 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"`**.

20) 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*.

21) 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*.

22) 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.

23) 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*.

24) 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*.

25) 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.

26) 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.

27) 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.

28) 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"`**.

29) 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*.

30) 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*.

31) 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.

32) 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*.

33) 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*.

34) 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.

35) 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.

36) Specifications for Feature X060, “XMLParse: Character string input and CONTENT option”:

a) Subclause 6.13, “<XML parse>”:

i) Without Feature X060, “XMLParse: Character string input and CONTENT option”, in conforming SQL language, the declared type of the <string value expression> immediately contained in <XML parse> shall not be a character string type and <XML parse> shall not immediately contain a <document or content> that is CONTENT.

37) Specifications for Feature X061, “XMLParse: Character string input and DOCUMENT option”:

a) Subclause 6.13, “<XML parse>”:

i) Without Feature X061, “XMLParse: Character string input and DOCUMENT option”, in conforming SQL language, the declared type of the <string value expression> immediately contained in <XML parse> shall not be a character string type and <XML parse> shall not immediately contain a <document or content> that is DOCUMENT.

38) Specifications for Feature X065, “XMLParse: BLOB input and CONTENT option”:

a) Subclause 6.13, “<XML parse>”:

i) Without Feature X065, “XMLParse: BLOB input and CONTENT option”, in conforming SQL language, the declared type of the <string value expression> immediately contained in <XML parse> shall not be a binary string type and <XML parse> shall not immediately contain a <document or content> that is CONTENT.

39) Specifications for Feature X066, “XMLParse: BLOB input and DOCUMENT option”:

a) Subclause 6.13, “<XML parse>”:

i) Without Feature X066, “XMLParse: BLOB input and DOCUMENT option”, in conforming SQL language, the declared type of the <string value expression> immediately contained in <XML parse> shall not be a binary string type and <XML parse> shall not immediately contain a <document or content> that is DOCUMENT.

40) Specifications for Feature X070, “XMLSerialize: Character string serialization and CONTENT option”:

a) Subclause 6.6, “<string value function>”:

i) Insert this CR Without Feature X070, “XMLSerialize: Character string serialization and CONTENT option”, conforming SQL language shall not contain an <XML character string serialization> that immediately contains a <document or content or sequence> that is CONTENT.

41) Specifications for Feature X071, “XMLSerialize: Character string serialization and DOCUMENT option”:

a) Subclause 6.6, “<string value function>”:

i) Insert this CR Without Feature X071, “XMLSerialize: Character string serialization and DOCUMENT option”, conforming SQL language shall not contain an <XML character string serialization> that immediately contains a <document or content or sequence> that is DOCUMENT.

42) Specifications for Feature X072, “XMLSerialize: Character string serialization and SEQUENCE option”:

a) Subclause 6.6, “<string value function>”:

i) Insert this CR Without Feature X072, “XMLSerialize: Character string serialization and SEQUENCE option”, conforming SQL language shall not contain an <XML character string serialization> that immediately contains a <document or content or sequence> that is SEQUENCE.

43) Specifications for Feature X073, “XMLSerialize: BLOB serialization and CONTENT option”:

a) Subclause 6.6, “<string value function>”:

i) Insert this CR Without Feature X073, “XMLSerialize: BLOB serialization and CONTENT option”, conforming SQL language shall not contain an <XML binary string serialization> that immediately contains a <document or content or sequence> that is CONTENT.

44) Specifications for Feature X074, “XMLSerialize: BLOB serialization and DOCUMENT option”:

a) Subclause 6.6, “<string value function>”:

i) Insert this CR Without Feature X074, “XMLSerialize: BLOB serialization and DOCUMENT option”, conforming SQL language shall not contain an <XML binary string serialization> that immediately contains a <document or content or sequence> that is DOCUMENT.

45) Specifications for Feature X075, “XMLSerialize: BLOB serialization and SEQUENCE option”:

a) Subclause 6.6, “<string value function>”:

i) Insert this CR Without Feature X075, “XMLSerialize: BLOB serialization and SEQUENCE option”, conforming SQL language shall not contain an <XML binary string serialization> that immediately contains a <document or content or sequence> that is SEQUENCE.

46) Specifications for Feature X076, “XMLSerialize: VERSION”:

a) Subclause 6.6, “<string value function>”:

i) Insert this CR Without Feature X076, “XMLSerialize: VERSION”, in conforming SQL language, <XML character string serialization> shall not contain VERSION.

ii) Insert this CR Without Feature X076, “XMLSerialize: VERSION”, in conforming SQL language, <XML binary string serialization> shall not contain VERSION.

47) Specifications for Feature X080, “Namespaces in XML publishing”:

a) Subclause 6.11, "<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 6.12, "<XML forest>":

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

48) Specifications for Feature X081, "Query-level XML namespace declarations":

a) Subclause 7.2, "<query expression>":

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

49) Specifications for Feature X082, "XML namespace declarations in DML":

a) Subclause 14.1, "<delete statement: searched>":

i) 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 lexically scoped options> that contains an <XML namespace declaration>.

b) Subclause 14.2, "<insert statement>":

i) Insert this CR Without Feature X082, "XML namespace declarations in DML", in conforming SQL language, an <insert statement> shall not immediately contain an <XML lexically scoped options> that contains an <XML namespace declaration>.

c) Subclause 14.3, "<merge statement>":

i) Insert this CR Without Feature X082, "XML namespace declarations in DML", in conforming SQL language, a <merge statement> shall not immediately contain an <XML lexically scoped options> that contains an <XML namespace declaration>.

d) Subclause 14.4, "<update statement: positioned>":

i) 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 lexically scoped options> that contains an <XML namespace declaration>.

e) Subclause 14.5, "<update statement: searched>":

i) 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 lexically scoped options> that contains an <XML namespace declaration>.

50) Specifications for Feature X083, "XML namespace declarations in DDL":

a) Subclause 12.1, "<column definition>":

i) Insert this CR Without Feature X083, "XML namespace declarations in DDL", in conforming SQL language, a <generation expression> shall not immediately contain an <XML lexically scoped options> that contains an <XML namespace declaration>.

b) Subclause 12.2, "<check constraint definition>":

i) 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 lexically scoped options> that contains an <XML namespace declaration>.

c) Subclause 12.4, "<assertion definition>":

i) Insert this CR Without Feature X083, "XML namespace declarations in DDL", in conforming SQL language, an <assertion definition> shall not immediately contain an <XML lexically scoped options> that contains an <XML namespace declaration>.

51) Specifications for Feature X084, "XML namespace declarations in compound statements":

a) Subclause 15.1, "<compound statement>":

i) 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 lexically scoped options> that contains an <XML namespace declaration>.

52) Specifications for Feature X090, "XML document predicate":

a) Subclause 8.3, "<XML document predicate>":

i) Without Feature X090, "XML document predicate", conforming SQL language shall not contain <XML document predicate>.

53) Specifications for Feature X091, "XML content predicate":

a) Subclause 8.2, "<XML content predicate>":

i) Without Feature X091, "XML content predicate", conforming SQL language shall not contain <XML content predicate>.

54) Specifications for Feature X095, "XMLTable":

a) Subclause 7.1, "<table reference>":

i) Insert this CR Without Feature X095, "XMLTable", conforming SQL language shall not contain <XML table>.

55) Specifications for Feature X096, "XMLExists":

a) Subclause 8.4, "<XML exists predicate>":

i) Without Feature X096, "XMLExists", conforming SQL language shall not contain <XML exists predicate>.

56) Specifications for Feature X100, "Host language support for XML: CONTENT option":

a) Subclause 12.7, "<SQL-invoked routine>":

i) 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) 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) 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) 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) 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.

57) 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) 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) 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) 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.

58) Specifications for Feature X110, "Host language support for XML: VARCHAR mapping":

a) Subclause 12.7, "<SQL-invoked routine>":

i) 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) 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) 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>.

59) Specifications for Feature X111, "Host language support for XML: CLOB mapping":

a) Subclause 12.7, "<SQL-invoked routine>":

i) 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) 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) 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) 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>.

60) 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 an XML type or a distinct type whose source type is an XML type.

61) 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 an XML type or a distinct type whose source type is an XML type.

62) Specifications for Feature X131, "Query-level XMLBINARY clause":

a) Subclause 7.2, "<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 lexically scoped options> that contains an <XML binary encoding>.

63) Specifications for Feature X132, "XMLBINARY clause in DML":

a) Subclause 14.1, "<delete statement: searched>":

i) Insert this CR Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, a <delete statement: searched> shall not immediately contain an <XML lexically scoped options> that contains an <XML binary encoding>.

b) Subclause 14.2, "<insert statement>":

i) Insert this CR Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, an <insert statement> shall not immediately contain an <XML lexically scoped options> that contains an <XML binary encoding>.

c) Subclause 14.3, "<merge statement>":

i) Insert this CR Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, a <merge statement> shall not immediately contain an <XML lexically scoped options> that contains an <XML binary encoding>.

d) Subclause 14.4, "<update statement: positioned>":

i) Insert this CR Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, an <update statement: positioned> shall not immediately contain an <XML lexically scoped options> that contains an <XML binary encoding>.

e) Subclause 14.5, "<update statement: searched>":

i) Insert this CR Without Feature X132, "XMLBINARY clause in DML", in conforming SQL language, an <update statement: searched> shall not immediately contain an <XML lexically scoped options> that contains an <XML binary encoding>.

64) Specifications for Feature X133, "XMLBINARY clause in DDL":

a) Subclause 12.1, "<column definition>":

i) Insert this CR Without Feature X133, "XMLBINARY clause in DDL", in conforming SQL language, a <generation expression> shall not immediately contain an <XML lexically scoped options> that contains an <XML binary encoding>.

b) Subclause 12.2, "<check constraint definition>":

i) Insert this CR Without Feature X133, "XMLBINARY clause in DDL", in conforming SQL language, a <check constraint definition> shall not immediately contain an <XML lexically scoped options> that contains an <XML binary encoding>.

c) Subclause 12.4, "<assertion definition>":

i) Insert this CR Without Feature X133, "XMLBINARY clause in DDL", in conforming SQL language, an <assertion definition> shall not immediately contain an <XML lexically scoped options> that contains an <XML binary encoding>.

65) Specifications for Feature X134, "XMLBINARY clause in compound statements":

a) Subclause 15.1, "<compound statement>":

i) Insert this CR Without Feature X134, "XMLBINARY clause in compound statements", in conforming SQL language, a <compound statement> shall not immediately contain an <XML lexically scoped options> that contains an <XML binary encoding>.

66) Specifications for Feature X135, "XMLBINARY clause in subqueries":

a) Subclause 7.2, "<query expression>":

i) Insert this CR Without Feature X135, "XMLBINARY clause in subqueries", in conforming SQL language, a <subquery> shall not contain an <XML lexically scoped options> that contains an <XML binary encoding>.

67) Specifications for Feature X141, "IS VALID predicate: data-driven case":

a) Subclause 8.5, "<XML valid predicate>":

i) Without Feature X141, "IS VALID predicate: data-driven case", conforming SQL language shall not contain an <XML valid predicate> that does not contain <XML valid according to clause>.

68) Specifications for Feature X142, "IS VALID predicate: ACCORDING TO clause":

a) Subclause 8.5, "<XML valid predicate>":

i) Without Feature X142, "IS VALID predicate: ACCORDING TO clause", conforming SQL language shall not contain an <XML valid predicate> that contains <XML valid according to clause>.

69) Specifications for Feature X143, "IS VALID predicate: ELEMENT clause":

a) Subclause 8.5, "<XML valid predicate>":

i) Without Feature X143, "IS VALID predicate: ELEMENT clause", conforming SQL language shall not contain an <XML valid predicate> that contains <XML valid element clause>.

70) Specifications for Feature X144, "IS VALID predicate: schema location":

a) Subclause 8.5, "<XML valid predicate>":

i) Without Feature X144, "IS VALID predicate: schema location", conforming SQL language shall not contain an <XML valid predicate> that does not contain <XML valid schema location>.

71) Specifications for Feature X145, "IS VALID predicate outside check constraints":

a) Subclause 8.5, "<XML valid predicate>":

i) Without Feature X145, "IS VALID predicate outside check constraints", conforming SQL language shall not contain an <XML valid predicate> that is not directly contained in the <search condition> of a <check constraint definition>.

72) Specifications for Feature X151, “IS VALID predicate: without identity constraints”:

a) Subclause 8.5, “<XML valid predicate>”:

i) Without Feature X151, “IS VALID predicate: without identity constraints”, conforming SQL language shall not contain an <XML valid predicate> that contains or implies WITHOUT IDENTITY CONSTRAINTS.

73) Specifications for Feature X152, “IS VALID predicate: global identity contraints”:

a) Subclause 8.5, “<XML valid predicate>”:

i) Without Feature X152, “IS VALID predicate: global identity contraints”, conforming SQL language shall not contain an <XML valid predicate> that contains WITHOUT IDENTITY CONSTRAINTS GLOBAL.

74) Specifications for Feature X153, “IS VALID predicate: local identity constraints”:

a) Subclause 8.5, “<XML valid predicate>”:

i) Without Feature X153, “IS VALID predicate: local identity constraints”, conforming SQL language shall not contain an <XML valid predicate> that contains WITHOUT IDENTITY CONSTRAINTS LOCAL.

75) Specifications for Feature X154, “IS VALID predicate: DOCUMENT clause”:

a) Subclause 8.5, “<XML valid predicate>”:

i) Without Feature X154, “IS VALID predicate: DOCUMENT clause”, conforming SQL language shall not contain an <XML valid predicate> that that contains an <XML valid identity constraint option> that is DOCUMENT.

76) Specifications for Feature X160, “Information Schema for registered XML Schemas”:

a) Subclause 20.5, “XML_SCHEMA_ELEMENTS view”:

i) Without Feature X160, “Information Schema for registered XML Schemas”, conforming SQL languages shall not reference INFORMATION_SCHEMA.XML_SCHEMA_ELEMENTS.

b) Subclause 20.6, “XML_SCHEMA_NAMESPACES view”:

i) Without Feature X160, “Information Schema for registered XML Schemas”, conforming SQL languages shall not reference INFORMATION_SCHEMA.XML_SCHEMA_NAMESPACES.

c) Subclause 20.7, “XML_SCHEMAS view”:

i) Without Feature X160, “Information Schema for registered XML Schemas”, conforming SQL languages shall not reference INFORMATION_SCHEMA.XML_SCHEMAS.

d) Subclause 20.8, “Short name views”:

i) Insert this CR Without Feature X160, “Information Schema for registered XML Schemas”, conforming SQL languages shall not reference INFORMATION_SCHEMA.XML_SCH_ELEMENTS_S.

ii) Insert this CR Without Feature X160, “Information Schema for registered XML Schemas”, conforming SQL languages shall not reference INFORMATION_SCHEMA.XML_SCH_NSPACES_S.

iii) Insert this CR Without Feature X160, “Information Schema for registered XML Schemas”, conforming SQL languages shall not reference INFORMATION_SCHEMA.XML_SCHEMAS_S.

77) Specifications for Feature X170, “XML null handling options”:

a) Subclause 6.11, “<XML element>”:

i) Without Feature X170, “XML null handling options”, conforming SQL language shall not specify <XML content option>.

78) Specifications for Feature X171, “NIL ON NO CONTENT option”:

a) Subclause 6.11, “<XML element>”:

i) Without Feature X171, “NIL ON NO CONTENT option”, conforming SQL language shall not specify NIL ON NO CONTENT.

79) Specifications for Feature X181, “XML(UNTYPED DOCUMENT) type”:

a) Subclause 6.1, “<data type>”:

i) Insert this CR Without Feature X181, “XML(UNTYPED DOCUMENT) type”, conforming SQL language shall not contain an <XML type> whose <XML type modifier> is UNTYPED DOCUMENT.

b) Subclause 8.3, “<XML document predicate>”:

i) Without Feature X181, “XML(UNTYPED DOCUMENT) type”, in conforming SQL language, an <XML document predicate> shall not contain <XML untyped or any> that is UNTYPED.

80) Specifications for Feature X182, “XML(ANY DOCUMENT) type”:

a) Subclause 6.1, “<data type>”:

i) Insert this CR Without Feature X182, “XML(ANY DOCUMENT) type”, conforming SQL language shall not contain an <XML type> whose <XML type modifier> is ANY DOCUMENT.

b) Subclause 8.3, “<XML document predicate>”:

i) Without Feature X182, “XML(ANY DOCUMENT) type”, in conforming SQL language, an <XML document predicate> shall not contain <XML untyped or any> that is ANY.

81) Specifications for Feature X190, “XML(SEQUENCE) type”:

a) Subclause 6.1, “<data type>”:

i) Insert this CR Without Feature X190, “XML(SEQUENCE) type”, conforming SQL language shall not contain an <XML type> whose <XML type modifier> is SEQUENCE.

82) Specifications for Feature X201, “XMLQuery: RETURNING CONTENT”:

a) Subclause 6.15, “<XML query>”:

i) Without Feature X201, “XMLQuery: RETURNING CONTENT”, conforming SQL language shall not contain an <XML query> that contains RETURNING CONTENT.

83) Specifications for Feature X202, “XMLQuery: RETURNING SEQUENCE”:

a) Subclause 6.15, “<XML query>”:

i) Without Feature X202, “XMLQuery: RETURNING SEQUENCE”, conforming SQL language shall not contain an <XML query> that contains RETURNING SEQUENCE.

84) Specifications for Feature X211, “XML 1.1 support”:

a) Subclause 6.11, “<XML element>”:

i) Without Feature X211, “XML 1.1 support”, in conforming SQL language, an <XML element name> shall be an XML 1.0 QName.

ii) Without Feature X211, “XML 1.1 support”, in conforming SQL language, an <XML attribute name> shall be an XML 1.0 QName.

b) Subclause 6.12, “<XML forest>”:

i) Without Feature X211, “XML 1.1 support”, in conforming SQL language, a <forest element name> shall be an XML 1.0 QName.

c) Subclause 6.14, “<XML PI>”:

i) Without Feature X211, “XML 1.1 support”, in conforming SQL language, a <XML PI target> shall be an XML 1.0 QName.

d) Subclause 6.15, “<XML query>”:

i) Without Feature X211, “XML 1.1 support”, in conforming SQL language, the value of the <XQuery expression> shall be an XQuery expression with XML 1.0 lexical rules.

ii) Without Feature X211, “XML 1.1 support”, in conforming SQL language, the <identifier> contained in an <XML query variable> shall be an XML 1.0 NCName.

e) Subclause 11.3, “<XML lexically scoped options>”:

i) Without Feature X211, “XML 1.1 support”, in conforming SQL language, each <XML namespace prefix> shall be an XML 1.0 NCName.

85) Specifications for Feature X221, “XML passing mechanism BY VALUE”:

a) Subclause 11.5, “<XML passing mechanism>”:

i) Without Feature X221, “XML passing mechanism BY VALUE”, conforming SQL language shall not contain an <XML passing mechanism> that is BY VALUE.

86) Specifications for Feature X222, “XML passing mechanism BY REF”:

a) Subclause 11.5, “<XML passing mechanism>”:

i) Without Feature X222, “XML passing mechanism BY REF”, conforming SQL language shall not contain an <XML passing mechanism> that is BY REF.

87) Specifications for Feature X231, “XML(UNTYPED CONTENT) type”:

a) Subclause 6.1, “<data type>”:

i) Insert this CR Without Feature X231, “XML(UNTYPED CONTENT) type”, conforming SQL language shall not contain an <XML type> whose <XML type modifier> is UNTYPED CONTENT.

b) Subclause 8.2, “<XML content predicate>”:

i) Without Feature X231, “XML(UNTYPED CONTENT) type”, in conforming SQL language, an <XML content predicate> shall not contain <XML untyped or any> that is UNTYPED.

88) Specifications for Feature X232, “XML(ANY CONTENT) type”:

a) Subclause 6.1, “<data type>”:

i) Insert this CR Without Feature X232, “XML(ANY CONTENT) type”, conforming SQL language shall not contain an <XML type> whose <XML type modifier> is ANY CONTENT.

b) Subclause 8.2, “<XML content predicate>”:

i) Without Feature X232, “XML(ANY CONTENT) type”, in conforming SQL language, an <XML content predicate> shall not contain <XML untyped or any> that is ANY.

89) Specifications for Feature X241, “Explicit RETURNING CONTENT in XML publishing”:

a) Subclause 6.9, “<XML comment>”:

i) Without Feature X241, “Explicit RETURNING CONTENT in XML publishing”, in Conforming SQL language, an <XML comment> shall not specify an <XML returning clause> that is RETURNING CONTENT.

b) Subclause 6.10, “<XML concatenation>”:

i) Without Feature X241, “Explicit RETURNING CONTENT in XML publishing”, in Conforming SQL language, an <XML concatenation> shall not specify an <XML returning clause> that is RETURNING CONTENT.

c) Subclause 6.11, “<XML element>”:

i) Without Feature X241, “Explicit RETURNING CONTENT in XML publishing”, in Conforming SQL language, an <XML element> shall not specify an <XML returning clause> that is RETURNING CONTENT.

d) Subclause 6.12, “<XML forest>”:

i) Without Feature X241, “Explicit RETURNING CONTENT in XML publishing”, in Conforming SQL language, an <XML forest> shall not specify an <XML returning clause> that is RETURNING CONTENT.

e) Subclause 6.14, “<XML PI>”:

i) Without Feature X241, “Explicit RETURNING CONTENT in XML publishing”, in Conforming SQL language, an <XML PI> shall not specify an <XML returning clause> that is RETURNING CONTENT.

f) Subclause 11.2, “<aggregate function>”:

i) Insert this CR Without Feature X241, “Explicit RETURNING CONTENT in XML publishing”, in Conforming SQL language, an <XML aggregate> shall not specify an <XML returning clause> that is RETURNING CONTENT.

90) Specifications for Feature X242, “RETURNING SEQUENCE in XML publishing”:

a) Subclause 6.9, “<XML comment>”:

i) Without Feature X242, “RETURNING SEQUENCE in XML publishing”, in Conforming SQL language, an <XML comment> shall not specify an <XML returning clause> that is RETURNING SEQUENCE.

b) Subclause 6.10, “<XML concatenation>”:

i) Without Feature X242, “RETURNING SEQUENCE in XML publishing”, in Conforming SQL language, an <XML concatenation> shall not specify an <XML returning clause> that is RETURNING SEQUENCE.

c) Subclause 6.11, “<XML element>”:

i) Without Feature X242, “RETURNING SEQUENCE in XML publishing”, in Conforming SQL language, an <XML element> shall not specify an <XML returning clause> that is RETURNING SEQUENCE.

d) Subclause 6.12, “<XML forest>”:

i) Without Feature X242, “RETURNING SEQUENCE in XML publishing”, in Conforming SQL language, an <XML forest> shall not specify an <XML returning clause> that is RETURNING SEQUENCE.

e) Subclause 6.14, “<XML PI>”:

i) Without Feature X242, “RETURNING SEQUENCE in XML publishing”, in Conforming SQL language, an <XML PI> shall not specify an <XML returning clause> that is RETURNING SEQUENCE.

f) Subclause 11.2, “<aggregate function>”:

i) Insert this CR Without Feature X242, “Explicit RETURNING SEQUENCE in XML publishing”, in Conforming SQL language, an <XML aggregate> shall not specify an <XML returning clause> that is RETURNING SEQUENCE.

91) Specifications for Feature X251, “Persistent XML values of XML(UNTYPED DOCUMENT) type”:

a) Subclause 12.1, “<column definition>”:

i) Insert this CR Without Feature X251, “Persistent XML values of XML(UNTYPED DOCUMENT) type”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(UNTYPED DOCUMENT) type or a distinct type whose source type is the XML(UNTYPED DOCUMENT) type.

b) Subclause 12.3, “<view definition>”:

i) Insert this CR Without Feature X251, “Persistent XML values of XML(UNTYPED DOCUMENT) type”, conforming SQL language shall not contain a <view definition> whose declared type is based on either the XML(UNTYPED DOCUMENT) type or a distinct type whose source type is the XML(UNTYPED DOCUMENT) type.

92) Specifications for Feature X252, “Persistent XML values of XML(ANY DOCUMENT) type”:

a) Subclause 12.1, “<column definition>”:

i) Insert this CR Without Feature X252, “Persistent XML values of XML(ANY DOCUMENT) type”, conforming SQL language shall not contain a <column definition> whose declared type

is based on either the XML(ANY DOCUMENT) type or a distinct type whose source type is the XML(ANY DOCUMENT) type.

b) Subclause 12.3, "<view definition>":

i) Insert this CR Without Feature X252, "Persistent XML values of XML(ANY DOCUMENT) type", conforming SQL language shall not contain a <view definition> whose declared type is based on either the XML(ANY DOCUMENT) type or a distinct type whose source type is the XML(ANY DOCUMENT) type.

93) Specifications for Feature X253, "Persistent XML values of XML(UNTYPED CONTENT) type":

a) Subclause 12.1, "<column definition>":

i) Insert this CR Without Feature X253, "Persistent XML values of XML(UNTYPED CONTENT) type", conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(UNTYPED CONTENT) type or a distinct type whose source type is the XML(UNTYPED CONTENT) type.

b) Subclause 12.3, "<view definition>":

i) Insert this CR Without Feature X253, "Persistent XML values of XML(UNTYPED CONTENT) type", conforming SQL language shall not contain a <view definition> whose declared type is based on either the XML(UNTYPED CONTENT) type or a distinct type whose source type is the XML(UNTYPED CONTENT) type.

94) Specifications for Feature X254, "Persistent XML values of XML(ANY CONTENT) type":

a) Subclause 12.1, "<column definition>":

i) Insert this CR Without Feature X254, "Persistent XML values of XML(ANY CONTENT) type", conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(ANY CONTENT) type or a distinct type whose source type is the XML(ANY CONTENT) type.

b) Subclause 12.3, "<view definition>":

i) Insert this CR Without Feature X254, "Persistent XML values of XML(ANY CONTENT) type", conforming SQL language shall not contain a <view definition> whose declared type is based on either the XML(ANY CONTENT) type or a distinct type whose source type is the XML(ANY CONTENT) type.

95) Specifications for Feature X255, "Persistent XML values of XML(SEQUENCE) type":

a) Subclause 12.1, "<column definition>":

i) Insert this CR Without Feature X255, "Persistent XML values of XML(SEQUENCE) type", conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(SEQUENCE) type or a distinct type whose source type is the XML(SEQUENCE) type.

b) Subclause 12.3, "<view definition>":

i) Insert this CR Without Feature X255, "Persistent XML values of XML(SEQUENCE) type", conforming SQL language shall not contain a <view definition> whose declared type is based on either the XML(SEQUENCE) type or a distinct type whose source type is the XML(SEQUENCE) type.

(Blank page)

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) Subclause 4.2.4, "Registered XML Schemas":

   a) XML Schemas are registered through implementation-defined means.

   b) Whether a registered XML Schema is identified by the combination of its target namespace URI and schema location URI, or by its target namespace URI alone, is implementation-defined.

   c) The <registered XML Schema name>s and schema location URIs of the built-in XML Schemas are implementation-defined.

2) Subclause 4.7.1, "Privileges":

   a) USAGE privileges on registered XML Schemas are granted or revoked by implementation-defined means.

3) Subclause 4.10.1, "Mapping SQL character sets to Unicode":

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

4) Subclause 4.10.3, "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 4.10.9, "Mapping Unicode to SQL character sets":

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

6) Subclause 6.1, "<data type>":

   a) Whether XML defaults to XML(UNTYPED CONTENT) or XML(ANY CONTENT) is implementation-defined.

7) Subclause 6.3, "<cast specification>":

   a) The result of CAST (EMPTY AS XML(UNTYPED CONTENT)) or CAST (EMPTY AS XML(ANY CONTENT)) is an XQuery document node with implementation-defined base-uri and document-uri properties.

8) Subclause 6.4, "<XML cast specification>":

a) The default encoding of binary strings (either base64 or hex) is implementation-defined.

9) Subclause 6.6, "<string value function>":

a) The <XML encoding name>s supported for <XML binary string serialization> are implementation-defined.

b) If an <XML encoding specification> is not specified, then the implicit <XML encoding name> is implementation-defined.

c) When serializing an XML value, the default XML version is implementation-defined.

10) Subclause 6.9, "<XML comment>":

a) The function used to convert the content of a comment to Unicode is implementation-defined.

b) When invoking XMLComment with the RETURNING CONTENT option, the base-uri and document-uri of the generated XQuery document node is implementation-defined.

11) Subclause 6.10, "<XML concatenation>":

a) The base-uri and document-uri of any XQuery document node generated using the RETURNING CONTENT option is implementation-defined.

12) Subclause 6.11, "<XML element>":

a) The default encoding of binary strings (either in base64 or in hex) is implementation-defined.

b) If RETURNING CONTENT is specified or implied, then it is implementation-defined whether the declared type of <XML element> is XML(UNTYPED CONTENT) or XML(ANY CONTENT).

13) Subclause 6.12, "<XML forest>":

a) The default encoding of binary strings (either in base64 or in hex) is implementation-defined.

b) If RETURNING CONTENT is specified or implied, then it is implementation-defined whether the declared type of <XML forest> is XML(UNTYPED CONTENT) or XML(ANY CONTENT).

14) Subclause 6.14, "<XML PI>":

a) The function used to convert the content of a processing instruction to Unicode is implementation-defined.

b) The base-uri of the generated XQuery processing instruction node is implementation-defined.

c) When invoking XMLPI with the RETURNING CONTENT option, the base-uri and document-uri of the generated XQuery document node is implementation-defined.

15) Subclause 6.15, "<XML query>":

a) Which optional features of XQuery are supported is implementation-defined.

16) Subclause 8.5, "<XML valid predicate>":

a) If the SQL-implementation supports schema locations for registered XML Schemas, <XML valid according to URI> is specified, and <XML valid schema location> is not specified, then the schema location is chosen by a deterministic implementation-defined algorithm that is repeatable in the sense that if the algorithm is reevaluated with the same collection of registered XML Schemas for which the user has USAGE privilege, then the same schema location will be chosen.

b) If the <XML valid predicate> does not specify <XML valid according to clause>, then the registered XML Schema used to assess validity of a top-level element *E* of the XML value is chosen from among those registered XML Schemas for which the user has USAGE privilege and that have a global element declaration schema component whose namespace is the namespace of *E*, using a deterministic implementation-defined algorithm that is repeatable, in the sense that if the algorithm is re-evaluated with the same collection of registered XML Schemas that are accessible to the user and the same element *E*, then the same registered XML Schema will be chosen.

c) When validating an element or attribute *EOA* whose [namespace name] property is not equivalent to a namespace of the registered XML Schema being used to validate the element that is the parent or owner of *EOA*, a registered XML Schema is chosen for which the current user has USAGE privilege and whose target namespace is identical to the [namespace name] property of EOA, as defined by [Namespaces], using a deterministic implementation-defined algorithm that is repeatable, in the sense that if the algorithm is re-evaluated with the same collection of registered XML Schemas that are accessible to the user and the same element or attribute, then the same registered XML Schema will be chosen.

17) 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.

18) Subclause 9.3, "Mapping an SQL table to XML and an XML Schema document":

a) If the SQL-implementation supports Feature X211, "XML 1.1 support", then the version of XML that is generated may be implementation-defined.

b) The encoding of the results is implementation-defined.

c) The presence and value of the **`schemaLocation`** hint of the **`xsd:import`** in the generated XML Schema document is implementation-defined.

19) Subclause 9.4, "Mapping an SQL schema to an XML document and an XML Schema document":

a) If the SQL-implementation supports Feature X211, "XML 1.1 support", then the version of XML that is generated may be implementation-defined.

b) The encoding of the results is implementation-defined.

c) The presence and value of the **`schemaLocation`** hint of the **`xsd:import`** in the generated XML Schema document is implementation-defined.

20) Subclause 9.5, "Mapping an SQL catalog to an XML document and an XML Schema document":

a) If the SQL-implementation supports Feature X211, "XML 1.1 support", then the version of XML that is generated may be implementation-defined.

b) The encoding of the results is implementation-defined.

c) The presence and value of the **`schemaLocation`** hint of the **`xsd:import`** in the generated XML Schema document is implementation-defined.

21) Subclause 9.6, "Mapping an SQL table to XML Schema data types":

a) All annotations are implementation-defined.

22) Subclause 9.7, "Mapping an SQL schema to XML Schema data types":

a) All annotations are implementation-defined.

23) Subclause 9.8, "Mapping an SQL catalog to XML Schema data types":

a) All annotations are implementation-defined.

24) 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.

25) 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.

26) Subclause 10.11, "Construction of an XML element":

a) The base-uri and document-uri of any XQuery document node generated using the RETURNING CONTENT option is implementation-defined.

b) Whether the unparsed-entities property is copied from the XQuery document node of a nested XML value to the XQuery document node of the result using the RETURNING CONTENT option is implementation-defined.

27) Subclause 10.14, "Serialization of an XML value":

a) The encoding name for any character set other than UTF8, UTF16, or UTF32 is implementation-defined.

b) The encoding name for any encoding other than UTF8, UTF16, or UTF32 is implementation-defined.

c) The value of the standalone property in the XML declaration of the output is implementation-defined.

28) Subclause 10.15, "Parsing a string as an XML value":

a) Support for the [notations] property and the [unparsed entities] property of the XML document information item is implementation-defined.

b) Support for external DTDs is implementation-defined.

29) Subclause 10.19, "Creation of an XQuery expression context":

a) The XQuery static and dynamic context may be initialized with implementation-defined values, as permitted by [XQuery].

30) Subclause 10.20, "Determination of an XQuery formal type notation":

a) The XQuery formal type notation of an XQuery variable may denote an implementation-defined subtype of the type specified in this Subclause.

31) 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.

32) Subclause 18.4, "<embedded SQL COBOL program>":

a) The implicit character set in a <COBOL XML CLOB variable> is implementation-defined.

33) Subclause 18.5, "<embedded SQL Fortran program>":

a) The implicit character set in a <Fortran XML CLOB variable> is implementation-defined.

34) 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.

35) Subclause 20.1, "NCNAME domain":

a) It is implementation-defined whether the NCNAME domain specifies all variable-length character string values that conform to the rules for formation and representation of an XML 1.0 or XML 1.1 NCName.

(Blank page)

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 4.10.7, "Mapping an SQL schema to XML":

   a) The repeatable ordering of tables in an SQL schema is implementation-dependent.

2) 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.

3) 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.

4) Subclause 10.14, "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 character string serialization>, or <XML binary string serialization>, respectively, and with the PRESERVE WHITESPACE option, will yield a value identical to the original value.

5) Subclause 10.15, "Parsing a string as an XML value":

   a) The [base URI] property of all XML information items is implementation-dependent.

6) 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.

(Blank page)

Annex D

(informative)

Incompatibilities with ISO/IEC 9075:2003

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) In ISO/IEC 9075-13:2003, the <XML whitespace option> of <XML parse> was optional; in this edition of ISO/IEC 9075-14, <XML whitespace option> is no longer optional.

2) A number of additional <reserved word>s have been added to the language. These <reserved word>s are:

   — ABSENT

   — EMPTY

   — NIL

   — XMLBINARY

   — XMLCAST

   — XMLCOMMENT

   — XMLEXISTS

   — XMLPI

   — XMLQUERY

   — XMLTABLE

3) This edition of ISO/IEC 9075-14 has deleted <XML root>, which was part of ISO/IEC 9075-14:2003.

(Blank page)

Annex E

(informative)

SQL feature taxonomy

This Annex describes a taxonomy of features defined in this part of ISO/IEC 9075.

Table 16, “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 16 — 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	X034	XMLAgg

	Feature ID	Feature Name
12	X035	XMLAgg: ORDER BY option
13	X041	Basic table mapping: null absent
14	X042	Basic table mapping: null as nil
15	X043	Basic table mapping: table as forest
16	X044	Basic table mapping: table as element
17	X045	Basic table mapping: with target namespace
18	X046	Basic table mapping: data mapping
19	X047	Basic table mapping: metadata mapping
20	X048	Basic table mapping: base64 encoding of binary strings
21	X049	Basic table mapping: hex encoding of binary strings
22	X051	Advanced table mapping: null absent
23	X052	Advanced table mapping: null as nil
24	X053	Advanced table mapping: table as forest
25	X054	Advanced table mapping: table as element
26	X055	Advanced table mapping: with target namespace
27	X056	Advanced table mapping: data mapping
28	X057	Advanced table mapping: metadata mapping
29	X058	Advanced table mapping: base64 encoding of binary strings
30	X059	Advanced table mapping: hex encoding of binary strings
31	X060	XMLParse: Character string input and CONTENT option
32	X061	XMLParse: Character string input and DOCUMENT option
33	X065	XMLParse: BLOB input and CONTENT option
34	X066	XMLParse: BLOB input and DOCUMENT option
35	X070	XMLSerialize: Character string serialization and CONTENT option
36	X071	XMLSerialize: Character string serialization and DOCUMENT option

	Feature ID	Feature Name
37	X072	XMLSerialize: Character string serialization and SEQUENCE option
38	X073	XMLSerialize: BLOB serialization and CONTENT option
39	X074	XMLSerialize: BLOB serialization and DOCUMENT option
40	X075	XMLSerialize: BLOB serialization and SEQUENCE option
41	X076	XMLSerialize: VERSION
42	X080	Namespaces in XML publishing
43	X081	Query-level XML namespace declarations
44	X082	XML namespace declarations in DML
45	X083	XML namespace declarations in DDL
46	X084	XML namespace declarations in compound statements
47	X090	XML document predicate
48	X091	XML content predicate
49	X095	XMLTable
50	X096	XMLExists
51	X100	Host language support for XML: CONTENT option
52	X101	Host language support for XML: DOCUMENT option
53	X110	Host language support for XML: VARCHAR mapping
54	X111	Host language support for XML: CLOB mapping
55	X120	XML parameters in SQL routines
56	X121	XML parameters in external routines
57	X131	Query-level XMLBINARY clause
58	X132	XMLBINARY clause in DML
59	X133	XMLBINARY clause in DDL
60	X134	XMLBINARY clause in compound statements
61	X135	XMLBINARY clause in subqueries

	Feature ID	Feature Name
62	X141	IS VALID predicate: data-driven case
63	X142	IS VALID predicate: ACCORDING TO clause
64	X143	IS VALID predicate: ELEMENT clause
65	X144	IS VALID predicate: schema location
66	X145	IS VALID predicate outside check constraints
67	X151	IS VALID predicate: without identity constraints
68	X152	IS VALID predicate: global identity constraints
69	X153	IS VALID predicate: local identity constraints
70	X154	IS VALID predicate: DOCUMENT clause
71	X160	Information Schema for registered XML Schemas
72	X170	XML null handling options
73	X171	NIL ON NO CONTENT option
74	X181	XML(UNTYPED DOCUMENT) type
75	X182	XML(ANY DOCUMENT) type
76	X190	XML(SEQUENCE) type
77	X201	XMLQuery: RETURNING CONTENT
78	X202	XMLQuery: RETURNING SEQUENCE
79	X211	XML 1.1 support
80	X221	XML passing mechanism BY VALUE
81	X222	XML passing mechanism BY REF
82	X231	XML(UNTYPED CONTENT)
83	X232	XML(ANY CONTENT)
84	X241	Explicit RETURNING CONTENT in XML publishing
85	X242	RETURNING SEQUENCE in XML publishing
86	X251	Persistent XML values of XML(UNTYPED DOCUMENT) type

	Feature ID	Feature Name
87	**X252**	**Persistent XML values of XML(ANY DOCUMENT) type**
88	**X253**	**Persistent XML values of XML(UNTYPED CONTENT) type**
89	**X354**	**Persistent XML values of XML(ANY CONTENT) type**
90	**X255**	**Persistent XML values of XML(SEQUENCE) type**

Table 16, "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".

(Blank page)

Index

Index entries appearing in **boldface** indicate the page where the word, phrase, or BNF nonterminal was defined; index entries appearing in *italics* indicate a page where the BNF nonterminal was used in a Format; and index entries appearing in roman type indicate a page where the word, phrase, or BNF nonterminal was used in a heading, Function, Syntax Rule, Access Rule, General Rule, Leveling Rule, Table, or other descriptive text.

— D —

— E —

— F —

— G —

— H —

— I —

— J —

— K —

— L —

— M —

— N —

— O —

— P —

— R —

— S —

— T —

— U —

— V —

— W —

— X —

— Y —

— Z —