INTERNATIONAL STANDARD

ISO/IEC 9075-14

Fifth edition
2016-12-15

Information technology — Database languages — SQL —

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

Technologies de l'information — Langages de base de données — SQL —

Partie 14: Spécifications relatives au XML (SQL/XML)

Reference number
ISO/IEC 9075-14:2016(E)

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents

Tables

Table

Page

(Blank 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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, SC 32, *Data management and interchange*.

This fifth edition of ISO/IEC 9075-14 cancels and replaces the fourth edition (ISO/IEC 9075-14:2011), which has been technically revised. It also incorporates Technical Corrigenda ISO/IEC 9075-14:2011/Cor.1:2013 and ISO/IEC 9075-14:2011/Cor.2:2015.

A list of all parts in the ISO/IEC 9075 series, published under the general title *Information technology — Database languages — SQL*, can be found on the ISO website.

NOTE The individual parts of multi-part standards are not necessarily published together. New editions of one or more parts can be published without publication of new editions of other parts.

Introduction

The organization of ISO/IEC 9075-14 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 20, “Diagnostics management”, defines the diagnostics management facilities.

20) Clause 21, “Information Schema”, defines viewed tables that contain schema information.

21) Clause 22, “Definition Schema”, defines base tables on which the viewed tables containing schema information depend.

22) Clause 23, “The SQL/XML XML Schema”, defines the content of an XML namespace that is used when SQL and XML are utilized together.

23) Clause 24, “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 25, “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 the body of 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 the body of 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, “Deprecated features”, is an informative Annex. It lists features that the responsible Technical Committee intend will not appear in a future revised version of this part of ISO/IEC 9075.

29) Annex E, “Incompatibilities with ISO/IEC 9075:2011”, is an informative Annex. It lists incompatibilities with the previous version of this part of ISO/IEC 9075.

30) Annex F, “SQL feature taxonomy”, is an informative Annex. It identifies features 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.

31) Annex G, “Defect reports not addressed in this edition of this part of ISO/IEC 9075”, is an informative Annex. It describes the Defect Reports that were known at the time of publication of this part of this International Standard. Each of these problems is a problem carried forward from the previous edition of ISO/IEC 9075. No new problems have been created in the drafting of this edition of this International Standard.

In the text of this part of ISO/IEC 9075, Clauses and Annexes begin new odd-numbered pages, and in Clause 5, “Lexical elements”, through Clause 25, “Conformance”, Subclauses begin new pages. 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 ISO and IEC standards

[ISO9075-1] ISO/IEC 9075-1:2016, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*.

[ISO9075-2] ISO/IEC 9075-2:2016, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*.

[ISO9075-3] ISO/IEC 9075-3:2016, *Information technology — Database languages — SQL — Part 3: Call-Level Interface (SQL/CLI)*.

[ISO9075-4] ISO/IEC 9075-4:2016, *Information technology — Database languages — SQL — Part 4: Persistent Stored Modules (SQL/PSM)*.

[ISO9075-11] ISO/IEC 9075-11:2016, *Information technology — Database languages — SQL — Part 11: Information and Definition Schemas (SQL/Schemata)*.

[ISO10646] ISO/IEC 10646, *Information technology — Universal Multi-Octet Coded Character Set (UCS)*.

2.2 Other international standards

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

[Infoset] *(Recommendation) XML Information Set*.
http://www.w3.org/TR/xml-infoset

[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.

[Namespaces 1.0] *(Recommendation) Namespaces in XML 1.0*.
http://www.w3.org/TR/xml-names

[Namespaces 1.1] *(Recommendation) Namespaces in XML 1.1*.
http://www.w3.org/TR/xml-names11

[RFC3986] RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, T. Berners-Lee, R. Fielding, L. Masinter.
http://www.ietf.org/rfc/rfc3986.txt

[Schema1] *(Recommendation) XML Schema Part 1: Structures*.
http://www.w3.org/TR/xmlschema-1/

[Schema2] *(Recommendation) XML Schema Part 2: Datatypes*.
http://www.w3.org/TR/xmlschema-2/

[Serialization] *(Recommendation) XSLT 2.0 and XQuery 1.0 Serialization*.
http://www.w3.org/TR/xslt-xquery-serialization/

[Unicode] The Unicode Consortium, *The Unicode Standard*. (Information about the latest version of the Unicode standard can be found by using the "Latest Version" link on the "Enumerated Versions of The Unicode Standard" page.)
http://www.unicode.org/versions/enumeratedversions.html

[Unicode15] Davis, Mark and Dürst, Martin, *Unicode Standard Annex #15: Unicode Normalization Forms*. The Unicode Consortium.
http://www.unicode.org/reports/tr15/

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

[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. [UniXML] provides rules for character usage in XML.

[XML 1.0] *(Recommendation) Extensible Markup Language (XML) Version 1.0*.
http://www.w3.org/TR/xml

[XML 1.1] *(Recommendation) Extensible Markup Language (XML) Version 1.1*.
http://www.w3.org/TR/xml11

[XPath] *(Recommendation) XML Path Language (XPath) Version 2.0*.
http://www.w3.org/TR/xpath20

[XQuery] *(Recommendation) XQuery 1.0: an XML Query Language*.
http://www.w3.org/TR/xquery/

[XQueryDM] *(Recommendation) XQuery 1.0 and XPath 2.0 Data Model*.
http://www.w3.org/TR/xpath-datamodel/

[XQueryFO] *(Recommendation) XQuery 1.0 and XPath 2.0 Functions and Operators*.
http://www.w3.org/TR/xpath-functions/

[XQueryFS] *(Recommendation) XQuery 1.0 and XPath 2.0 Formal Semantics*.
http://www.w3.org/TR/xquery-semantics/

[XQueryUpdate] *(Candidate Recommendation)* XQuery Update Facility 1.0, W3C Working Draft.
http://www.w3.org/TR/xquery-update-10/

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 document, 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 document, the definitions of the following terms given in [Schema1] and/or [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 document, the following definitions apply:

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

3.1.3.2 canonical XML Schema literal
canonical lexical representation for an XML Schema type ***XST***, as defined in [Schema2], based on [CanonicalXML]

NOTE 2 — There is a unique canonical XML Schema literal for each value in the value space of ***XST***.

3.1.3.3 empty XQuery sequence
empty sequence, as defined in [XQueryDM]

3.1.3.4 global element declaration schema component
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 [ISO9075-2] and other parts of this International Standard

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

3.1.3.7 SQL value space
set of all values for a particular SQL <data type>

3.1.3.8 URI
Uniform Resource Identifier as defined in [RFC3986]

3.1.3.9 valid XML character
if Feature X211, "XML 1.1 support", is supported, then a valid XML 1.1 character; otherwise, a valid XML 1.0 character

3.1.3.10 valid XML 1.0 character
legal character as defined in [XML 1.0], rule [2], "Char"

3.1.3.11 valid XML 1.1 character
legal character as defined in [XML 1.1], rule [2], "Char"

3.1.3.12 XML attribute
attribute as defined by [XML]

3.1.3.13 XML attribute information item
attribute information item, as defined in [Infoset]

3.1.3.14 XML character information item
character information item, as defined in [Infoset]

3.1.3.15 XML declaration
XMLDecl, as defined in [XML]

3.1.3.16 XML document information item
document information item, as defined in [Infoset]

3.1.3.17 XML element
element as defined by [XML]

3.1.3.18 XML element information item
element information item, as defined in [Infoset]

3.1.3.19 XML information item

information item, as defined in [Infoset]

3.1.3.20 XML 1.0 Name
Name as defined by [XML 1.0]

3.1.3.21 XML 1.1 Name
Name as defined by [XML 1.1]

3.1.3.22 XML 1.0 NameChar
NameChar as defined by [XML 1.0]

3.1.3.23 XML 1.1 NameChar
NameChar as defined by [XML 1.1]

3.1.3.24 XML 1.1 NameStartChar
NameStartChar as defined by [XML 1.1]

3.1.3.25 XML namespace
XML namespace as defined by [Namespaces]

3.1.3.26 XML namespace prefix
namespace prefix as defined by [Namespaces]

3.1.3.27 XML 1.0 NCName
NCName, as defined by rule [4] in [Namespaces 1.0]

3.1.3.28 XML 1.1 NCName
NCName, as defined by rule [4] in [Namespaces 1.1]

3.1.3.29 XML 1.0 QName
QName, as defined by rule [7] of [Namespaces 1.0]

3.1.3.30 XML 1.1 QName
QName, as defined by rule [7] of [Namespaces 1.1]

3.1.3.31 XML 1.0 QName prefix
Prefix, as defined by rule [10] of [Namespaces 1.0]

3.1.3.32 XML 1.1 QName prefix
Prefix, as defined by rule [10] of [Namespaces 1.1]

3.1.3.33 XML 1.0 QName local part
LocalPart, as defined by rule [11] of [Namespaces 1.0]

3.1.3.34 XML 1.1 QName local part
LocalPart, as defined by rule [11] of [Namespaces 1.1]

3.1.3.35 XML Schema built-in data type
built-in datatype, as defined in [Schema2]

3.1.3.36 XML Schema complex type
complex type defined by a complex type definition, as defined in [Schema1]

3.1.3.37 XML Schema data type
datatype, as defined in [Schema2]

3.1.3.38 XML Schema document
schema document, as defined in [Schema1]

3.1.3.39 XML Schema primitive type
primitive type, as defined in [Schema2]

3.1.3.40 XML Schema simple type
simple type defined by a simple type definition, as defined in [Schema2]

3.1.3.41 XML Schema type
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

3.1.3.42 XML target namespace
target namespace as defined by [Schema1]

3.1.3.43 XML target namespace URI
URI of an XML target namespace

3.1.3.44 XML text
character string that is a substring of a textual XML 1.0 content or a textual XML 1.1 content, as defined in Subclause 10.16, "Parsing a string as an XML value"

3.1.3.45 XML value space
value space, as defined by [Schema2]

3.1.3.46 XQuery accessor
accessor, as defined in [XQueryDM]

3.1.3.47 XQuery atomic type
atomic type, as defined in [XQueryDM]

3.1.3.48 XQuery atomic value
atomic value, as defined in [XQueryDM]

3.1.3.49 XQuery attribute node
attribute node, as defined in [XQueryDM]

3.1.3.50 XQuery comment node
comment node, as defined in [XQueryDM]

3.1.3.51 XQuery datetime normalized value
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 3.3.2 "Dates and Times"

3.1.3.52 XQuery datetime timezone component
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 3.3.2, "Dates and Times"

3.1.3.53 XQuery document node
document node, as defined in [XQueryDM]

3.1.3.54 XQuery dynamic context
dynamic context, as defined in [XQuery]

3.1.3.55 XQuery element node
element node, as defined in [XQueryDM]

3.1.3.56 XQuery error
static error, type error or dynamic error, as defined in [XQuery]

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

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

3.1.3.59 XQuery expression
XQuery expression with XML 1.0 lexical rules, or an XQuery expression with XML 1.1 lexical rules

3.1.3.60 XQuery expression context
expression context, consisting of a static context and a dynamic context, as defined in [XQuery]

3.1.3.61 XQuery expression with XML 1.0 lexical rules
expression, as defined by rule [31], "Expr", in [XQuery], with rules [4], "NCName", and [7], "QName", interpreted to reference [Namespaces 1.0]

3.1.3.62 XQuery expression with XML 1.1 lexical rules
expression, as defined by rule [31], "Expr", in [XQuery], with rules [4], "NCName", and [7], "QName", interpreted to reference [Namespaces 1.1]

3.1.3.63 XQuery formal type notation
formal type notation, as defined by rule [24(Formal)] "Type", in [XQueryFS]

3.1.3.64 XQuery item
item, as defined in [XQueryDM]

3.1.3.65 XQuery namespace node
namespace node, as defined in [XQueryDM]

3.1.3.66 XQuery node
node, as defined in [XQueryDM]

3.1.3.67 XQuery node identity
node identity, as defined in [XQueryDM]

3.1.3.68 XQuery node property
property of a node, as defined in [XQueryDM]

3.1.3.69 XQuery processing instruction node
processing instruction node, as defined in [XQueryDM]

3.1.3.70 XQuery sequence
sequence, as defined in [XQueryDM]

3.1.3.71 XQuery serialization normalization
normalization, as defined in [Serialization] and [Unicode15]

3.1.3.72 XQuery static context
static context, as defined in [XQuery]

3.1.3.73 XQuery text node

text node, as defined in [XQueryDM]

3.1.3.74 XQuery tree
tree, as defined in [XQueryDM]

3.1.3.75 XQuery variable
variable, as defined in [XQuery]

3.2 Notation

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

Insert this paragraph Many of the standards referenced in Subclause 2.2, "Other international standards", are produced and maintained by the World Wide Web Consortium, including [XML], [XPath], [Namespaces], [Schema1], [Schema2], [Infoset], [XQuery]. W3C calls these international standards "Recommendations", and places them on its web site (http://www.w3.org). Whenever this part of ISO/IEC 9075 mentions a value, type, or other object (broadly conceived) that is specified by a W3C Recommendation, then that object is indicated using bold monospace font (for example, **`<xs:element>`**).

Insert this paragraph Similarly, when a textual variable in a Rule denotes an object that is specified by a W3C Recommendation, then the variable is written in italicized bold monospace font (for example, ***FACETP***).

Insert this paragraph The following list enumerates many, but not necessarily all, of the kinds of objects that are specified by W3C Recommendations:

— XML text.

— XML values.

— XML namespaces and XML namespace prefixes.

— XML Schema types.

— XQuery types.

— XQuery nodes.

— XQuery node properties.

— XQuery expressions.

— XQuery functions.

— XQuery variables.

— XQuery formal type notations.

— XQuery static and dynamic contexts.

Insert this paragraph On the other hand, an object that has some feature or aspect that is not specified by W3C is not in bold. For example, W3C objects do not have the ability to be null. Since a value of XML type may be null, it is not represented in bold. As another example, a registered XML Schema has an SQL privilege associated with it. In addition, non-bold variables are used for SQL BNF non-terminals, even when they are used to identify W3C objects.

Insert this paragraph This part of this International Standard specifies an interface between SQL and objects specified by W3C. As such, some values may be both W3C objects and also SQL objects. Therefore no notational convention can be totally consistent. In particular, it is permitted to assign a bold value to a non-bold variable, or to assign a non-bold value to a bold variable (provided the non-bold value has been determined to have no SQL-only feature, such as being a null). It is also permitted to compare a bold value and a non-bold value.

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.

Insert this paragraph In this part of this International Standard, <left bracket>/<right bracket> pairs occur in five contexts:

— As part of BNF productions, in which a pair of brackets surrounds one or more non-terminal and/or terminal symbols that are, treated as a single group, optional.

— In code examples, in which the brackets are literal characters that represent themselves as part of the code.

— In ordinary text, in which a pair of brackets enclose a word or phrase that identifies a publication referenced in Clause 2, “Normative references”.

— In ordinary text, in which a pair of brackets enclose a number that identifies the number of a BNF production specified in the publication whose reference appears nearby.

— In ordinary text, in which a pair of brackets enclose a word or phrase that identifies the name of a property defined in [Infoset].

Insert this paragraph In this part of this International Standard, <left brace>/<right brace> pairs occur in three contexts:

— As part of BNF productions, in which a pair of braces surrounds one or more non-terminal and/or terminal symbols that are to be handled as a single group.

— In code examples, in which the braces are literal characters that represent themselves as part of the code.

— In ordinary text, in which the braces enclose a word or phrase that identifies the name of a property defined in [Schema1].

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

Insert after 3rd paragraph The data types XML(DOCUMENT(UNTYPED)), XML(DOCUMENT(ANY)), XML(DOCUMENT(XMLSCHEMA)), XML(CONTENT(UNTYPED)), XML(CONTENT(ANY)), XML(CONTENT(XMLSCHEMA)), and XML(SEQUENCE) are referred to as the *XML types*. Values of XML types are called *XML values*.

4.1.2 Data type terminology

This Subclause modifies Subclause 4.1.4, "Data type terminology", in ISO/IEC 9075-2.

Augment the list in the 11th 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

4.2.1 Introduction

Fuller descriptions of XML concepts can be found in the documents about XML included in Clause 2, "Normative references".

4.2.2 XML types

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

— The name of the data type (XML).

— The primary XML type modifier described by the <primary XML type modifier> (DOCUMENT, CONTENT, or SEQUENCE).

— The secondary XML type modifier described by the <secondary XML type modifier> (UNTYPED, ANY, or XMLSCHEMA), if any.

— The registered XML Schema descriptor of the indicated registered XML Schema, if any.

— The XML namespace URI of the indicated XML namespace, if any.

— The XML NCName of the indicated global element declaration schema component, if any.

NOTE 3 — "Indicated registered XML Schema", "indicated XML namespace", and "indicated global element declaration schema component" are defined in Subclause 11.6, "<XML valid according to clause>".

An XML type whose primary XML type modifier is DOCUMENT and whose secondary XML type modifier is UNTYPED is called an XML(DOCUMENT(UNTYPED)) type.

An XML type whose primary XML type modifier is DOCUMENT and whose secondary XML type modifier is ANY is called an XML(DOCUMENT(ANY)) type.

An XML type whose primary XML type modifier is DOCUMENT and whose secondary XML type modifier is XMLSCHEMA is called an XML(DOCUMENT(XMLSCHEMA)) type.

NOTE 4 — The number of XML(DOCUMENT(XMLSCHEMA)) types is determined by the number of registered XML Schemas, the number of XML namespaces in those XML Schemas, and the number of global element declaration schema components in those XML Schemas.

An XML type whose primary XML type modifier is CONTENT and whose secondary XML type modifier is UNTYPED is called an XML(CONTENT(UNTYPED)) type.

An XML type whose primary XML type modifier is CONTENT and whose secondary XML type modifier is ANY is called an XML(CONTENT(ANY)) type.

An XML type whose primary XML type modifier is CONTENT and whose secondary XML type modifier is XMLSCHEMA is called an XML(CONTENT(XMLSCHEMA)) type.

NOTE 5 — The number of XML(CONTENT(XMLSCHEMA)) types is determined by the number of registered XML Schemas, the number of XML namespaces in those XML Schemas, and the number of global element declaration schema components in those XML Schemas.

An XML type whose primary XML type modifier is SEQUENCE is called an XML(SEQUENCE) type.

When determining the declared type of an <XML value expression>, an SQL-implementation is permitted to use the XML(DOCUMENT(ANY)) type in place of XML(DOCUMENT(UNTYPED)) and XML(DOCUMENT(XMLSCHEMA)) types, the XML(CONTENT(ANY)) type in place of XML(CONTENT(UNTYPED)), XML(CONTENT(XMLSCHEMA)), and XML(DOCUMENT(ANY)) types, and the XML(SEQUENCE) type in place of the XML(CONTENT(ANY)) type.

NOTE 6 — The above substitutions can be applied transitively. For instance, an SQL-implementation is permitted to use the XML(SEQUENCE) type in place of the XML(DOCUMENT(UNTYPED)) type.

4.2.3 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(CONTENT(ANY)).

Every XML value that is either:

— The null value.

— A non-null value of type XML(CONTENT(ANY)) that is an XQuery document node ***D*** such that both of the following are true:

- For every XQuery element node that is contained in the XQuery tree ***T*** rooted in ***D***, the **type-name** property is **`xs:untyped`** and the **nilled** property is **`false`**.
- For every XQuery attribute node that is contained in ***T***, the **type-name** property is **`xs:untypedAtomic`**.

is a value of type XML(CONTENT(UNTYPED)).

Every XML value that is either:

— The null value.

— A non-null value of type XML(CONTENT(ANY)) that is an XQuery document node ***D*** such that every XQuery element node that is contained in the XQuery tree ***T*** rooted in ***D*** is valid according to at least one of the following:

- An XML Schema ***S***.
- An XML namespace ***N*** in an XML Schema ***S***.
- A global element declaration schema component ***E*** in an XML Schema ***S***.

is a value of type XML(CONTENT(XMLSCHEMA)) whose type descriptor includes the registered XML Schema descriptor of ***S*** and, if ***N*** is specified, the XML namespace URI of ***N*** or, if ***E*** is specified, the XML namespace URI of ***E*** and the XML NCName of ***E***.

Every XML value that is either:

— The null value.

— A non-null value of type XML(CONTENT(ANY)) that is an XQuery document node whose **children** property has exactly one XQuery element node, zero or more XQuery comment nodes, and zero or more XQuery processing instruction nodes.

is a value of type XML(DOCUMENT(ANY)).

Every XML value that is either:

— The null value.

— A non-null value of type XML(CONTENT(UNTYPED)) that is an XQuery document node whose **children** property has exactly one XQuery element node, zero or more XQuery comment nodes, and zero or more XQuery processing instruction nodes.

is a value of type XML(DOCUMENT(UNTYPED)).

Every XML value that is either:

— The null value.

— A non-null value of type XML(DOCUMENT(ANY)) that is valid according to at least one of the following:

- An XML Schema ***S***.
- An XML namespace ***N*** in an XML Schema ***S***.
- A global element declaration schema component ***E*** in an XML Schema ***S***.

is a value of type XML(DOCUMENT(XMLSCHEMA)) whose type descriptor includes the registered XML Schema descriptor of ***S*** and, if ***N*** is specified, the XML namespace URI of ***N***; or, if ***E*** is specified, the XML namespace URI of ***E*** and the XML NCName of ***E***.

4.2.4 XML comparison and assignment

A value of an XML type *S* is assignable to a site of an XML type *T* if any of the following is true:

— *T* is either XML(DOCUMENT(UNTYPED)) or XML(CONTENT(UNTYPED)) and *S* is either XML(DOCUMENT(UNTYPED)) or XML(CONTENT(UNTYPED)).

— *T* is either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) and *S* is either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) such that the registered XML Schema descriptors included in the XML type descriptors of both *T* and *S* identify identical XML Schemas, the XML namespace URI included in the XML type descriptor of *T*, if any, is identical to the XML namespace URI included in the XML type descriptor of *S*, as defined by [Namespaces], and the XML NCName of the global element declaration schema component included in the XML type descriptor of *T*, if any, is identical to the XML NCName of the global element declaration schema component included in the XML type descriptor of *S*.

NOTE 7 — The notion of identical XML Schemas is defined in Subclause 4.2.6, “Registered XML Schemas”.

— *T* is either XML(DOCUMENT(ANY)), XML(CONTENT(ANY)), or XML(SEQUENCE) and *S* is an XML type.

Operations that involve assignment of XML values use the keywords BY REF to indicate that the assignment preserves XQuery node identity, and BY VALUE to indicate that the assignment loses XQuery node identity.

XML values are not comparable.

4.2.5 Operations involving XML values

<XML document> is an operator that returns an XML value, given another XML value. The new XML value consists of an XQuery document node that is constructed according to the rules of the computed document constructor in [XQuery].

<XML element> is an operator that returns an XML value, given an XML element name, an optional list of XML namespaces, 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. The result, an XQuery element node, may optionally be placed as the sole child of an XQuery document node.

<XML forest> is an operator that returns an XML value, given an optional list of XML namespaces and 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. The result, an XQuery sequence, may optionally be placed in an XQuery document node.

<XML concatenation> is an operator that returns an XML value by concatenating a list of XML values, as defined in Subclause 6.12, "<XML concatenation>". The result, an XQuery sequence, may optionally be placed in an XQuery document node.

<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 instruction 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 text> is an operator that returns an XML value, given a character string *CS*. The XML value consists of an XQuery text node whose **content** property is *CS*, mapped to Unicode. The XQuery text 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. The result is specified by means of <XML table row pattern>, <XML table argument list>, and <XML table column definitions>. The result is computed in the following steps:

— The <XML table row pattern> is an XQuery expression that is evaluated, with <XML table argument list> as the (optional) input arguments. This produces an intermediate result, an XQuery sequence.

— Each XQuery item in the intermediate result is used to create a row of the final result.

— The columns of the result are specified by <XML table column definitions>, which is a collection of <XML table column definition>s modeled on a <table definition>. Thus, each result column has a <data type>, and, optionally, a <default clause>. Each <XML table column definition> also specifies, explicitly or implicitly, an <XML table column pattern> *XTCP*, which is an XQuery expression.

— To generate the value for a particular row and column of the final result, the column pattern (*i.e.*, the XQuery expression *XTCP*) is evaluated using a particular XQuery item of the intermediate result as the

input parameter, and then cast to the <data type> of the result column. If *XTCP* evaluates to the empty XQuery sequence, then the column's <default clause>, if any, determines the column's default value.

<XML character string serialization> is an operator that returns a character string, given an XML value, using the algorithm in [Serialization]. Optionally, the input XML value may be checked to ensure that it is an XQuery document node whose **children** property has exactly one XQuery element node. Other options may be used to specify the version of XML text that is produced (*i.e.*, either a textual XML 1.0 content or a textual XML 1.1 content); and whether an XML declaration is produced or not.

<XML binary string serialization> is an operator that returns a binary string, given an XML value, using the algorithm in [Serialization]. <XML binary string serialization> supports the same options as <XML character string serialization>, plus the ability to specify the encoding (character set) of the result.

<XML parse> is an operator that parses a character string or binary string value, i.e., converts XML text to a set of XML information items, according to the rules of [Infoset], and then converts that set of XML information items into an XQuery document node, according to the rules of [XQueryDM].

<XML validate> is an operator that validates an XML value according to a registered XML Schema. Optionally, the registered XML Schema against which to validate may be chosen on the basis of the contents of the data value, or it may be specified directly in the <XML validate>. In the latter case, additional options allow the validation to be confined to a particular namespace of the designated registered XML Schema, or to a particular global element declaration schema component. Upon successful validation, <XML validate> returns a copy of the input XML value augmented with default values and type annotations. If validation fails, <XML validate> raises an exception.

<XML content predicate> is a predicate that determines if an XML value is an XQuery document node.

<XML document predicate> is a predicate that determines if an XML value is an XQuery document node whose **children** property contains exactly one XQuery element node, zero or more XQuery comment nodes, and zero or more XQuery processing instruction nodes.

<XML exists predicate> is a predicate that evaluates an XQuery expression and determines if the result is a null value, an empty XQuery sequence, or a non-empty XQuery sequence.

<XML valid predicate> is a predicate that determines if an XML value is valid according to a registered XML Schema. Optionally, the registered XML Schema to validate against may be chosen on the basis of the contents of the data value, or it may be specified directly in the <XML valid predicate>. In the latter case, additional options allow the validation to be confined to a particular namespace of the designated registered XML Schema, or to a particular global element declaration schema component.

4.2.6 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 8 — 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`**”.

An XML namespace *NS* contained in a registered XML Schema is *non-deterministic* if *NS* contains a global element declaration schema component that is non-deterministic.

A registered XML Schema is *non-deterministic* if it contains a non-deterministic XML namespace.

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 of whether the registered XML Schema is permanently registered.

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

— 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.

NOTE 9 — Without Feature X161, "Advanced Information Schema for registered XML Schemas", information whether an XML Schema is deterministic, information about the collection of namespaces defined in that XML Schema, and, for each such namespace information about the global element declaration schema components in that namespace, is not available in the XML_SCHEMAS, XML_SCHEMA_NAMESPACES, and XML_SCHEMA_ELEMENTS views.

A registered XML Schema is identified by its <registered XML Schema name>.

Two registered XML Schemas are considered *identical* if both are identified by the same <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`	implementation-defined	implementation-defined
`xsi`	`http://www.w3.org/2001/XMLSchema-instance`	implementation-defined	implementation-defined
`sqlxml`	`http://standards.iso.org/iso/9075/-2003/sqlxml`	implementation-defined	implementation-defined

NOTE 10 — 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 a <data type>, an <XML validate>, or an <XML valid predicate> that contains an <XML valid according to what> *XVACC* 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 that is referenced by *XVACC* 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. The same registered XML Schema is referenced whenever the view is used, or 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 [ISO9075-2] and the XML types can be specified by an <XML cast specification>.

A conversion from an XML type to a non-XML type is computed as follows:

1) XQuery document nodes are removed from the source value.

2) The result is converted to an XQuery sequence of XQuery atomic values ***AV*** using the XQuery function **`fn:data()`**.

3) An XML Schema data type ***XT*** is chosen to correspond to the target <data type>. For example, if the target <data type> is INTEGER then *XT* is **`xs:integer`**.

4) ***AV*** is converted to ***XT*** using the rules of [XQuery], producing ***BV***.

5) ***BV*** is a value of an XML Schema data type, which is either an XML Schema primitive type, or derived from an XML Schema primitive type. For example, ***BV*** might be a value of XML Schema type **`xs:short`**, which is derived from **`xs:decimal`**. The value spaces of most XML Schema primitive types are identified with an SQL value space. For example, the XML value space of **`xs:decimal`** is identified with the SQL value space of exact numeric values. If, say, ***BV*** has the value **`9075.14`** in the XML value space of **`xs:decimal`**, then ***BV*** is also regarded as being the value **`9075.14`** in the SQL value space of exact numeric values. These value space identifications are defined in Subclause 4.10.7, "Mapping XQuery atomic values to SQL values".

6) ***BV***, regarded now as a value of an SQL value space, is converted to the target <data type>, using a <cast specification>.

A conversion from a non-XML type to an XML type is computed as follows:

1) The source value is converted to a <character string literal> whose value is the result of applying the General Rules of Subclause 9.8, "Mapping values of SQL data types to values of XML Schema data types".

2) The <character string literal> is parsed using <XML parse>, producing an XQuery document node ***D***.

3) If the target <data type> is XML(CONTENT(UNTYPED)) or XML(CONTENT(ANY)), then ***D*** is the result.

4) If the target <data type> is XML(SEQUENCE), then ***D*** is converted to an XQuery atomic value by evaluating an XQuery cast expression. The XQuery type to convert to is chosen based on the declared type of the source value. For example, if the declared type of the source is exact numeric, then the XQuery type of the result is **`xs:decimal`**.

4.4 Data analysis operations (involving tables)

This Subclause modifies Subclause 4.16, "Data analysis operations (involving tables)", in ISO/IEC 9075-2.

4.4.1 Aggregate functions

This Subclause modifies Subclause 4.16.4, "Aggregate functions", in ISO/IEC 9075-2.

Augment 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 4.33, "SQL-invoked routines", in ISO/IEC 9075-2.

4.5.1 Routine descriptors

This Subclause modifies Subclause 4.33.5, "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 character string type descriptor of the associated string type.

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

4.6 SQL-statements

This Subclause modifies Subclause 4.39, "SQL-statements", in ISO/IEC 9075-2.

4.6.1 SQL-statements classified by function

This Subclause modifies Subclause 4.39.2, "SQL-statements classified by function", in ISO/IEC 9075-2.

4.6.1.1 SQL-session statements

This Subclause modifies Subclause 4.39.2.7, "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 4.40, "Basic security model", in ISO/IEC 9075-2.

4.7.1 Privileges

This Subclause modifies Subclause 4.40.2, "Privileges", in ISO/IEC 9075-2.

Augment the list following 1st paragraph

— registered XML Schema

Augment the list following 11th 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 4.43, "SQL-sessions", in ISO/IEC 9075-2.

4.8.1 SQL-session properties

This Subclause modifies Subclause 4.43.3, "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>.

Augment the 13th paragraph

— The current XML option.

4.9 XML namespaces

This part of ISO/IEC 9075 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
`xs`	`http://www.w3.org/2001/XMLSchema`
`xsi`	`http://www.w3.org/2001/XMLSchema-instance`
`sqlxml`	`http://standards.iso.org/iso/9075/2003/sqlxml`

A conforming implementation is not required to use the XML namespace prefixes **`xs`**, **`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 23, "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 23, "The SQL/XML XML Schema". This file has been created for the convenience of the implementors of this part of ISO/IEC 9075.

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, as defined in [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 11 — The entity references **<**, **&**, **>**, **'**, and **"**, as well as character references, 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 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*.

4.10.3 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 12 — 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.

4.10.4 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 9.3, "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 13 — This part of this International Standard specifies no syntax for invoking the mapping specified in Subclause 9.3, "Mapping XML Names to SQL <identifier>s". This specification is intended to be used by applications and referenced by other standards. It is the responsibility of any such application or other standard to ensure that the correct number of arguments as well as a valid value for each argument are supplied for this mapping.

NOTE 14 — 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>.

4.10.5 Mapping SQL data types to XML

For each SQL type or domain that is not unmappable, there is a corresponding XML Schema type. The mapping is fully specified in Subclause 9.5, "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 **xs: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 **xs:length** is used. Otherwise (*i.e.*, CHARACTER when the mapping is not homomorphic, as well as CHARACTER VARYING and CHARACTER LARGE OBJECT), the facet **xs: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 **xs:hexBinary** or the XML Schema type **xs:base64Binary**. The **xs:maxLength** facet is set to the maximum length of the binary string in octets. Annotations optionally indicate the SQL type (BINARY, BINARY VARYING, BINARY LARGE OBJECT) and the maximum length in octets. For <XML element> and <XML forest>, the choice of whether to map to **xs:hexBinary** or **xs: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 9.11, "Mapping an SQL table to XML and an XML Schema document", Subclause 9.14, "Mapping an SQL schema to an XML document and an XML Schema document", and Subclause 9.17, "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 **`xs:decimal`** using the facets **`xs:precision`** and **`xs:scale`**. It is implementation-defined whether the SQL types INTEGER, SMALLINT, and BIGINT are mapped to the XML Schema type **`xs:integer`** using the facets **`xs:maxInclusive`** and **`xs:minInclusive`** or to the closest XML Schema type that is a subtype of **`xs:integer`**, using the facets **`xs:maxInclusive`** and **`xs: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 **`xs: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 **`xs: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 **`xs:boolean`**. Optionally, an annotation indicates the SQL type (BOOLEAN).

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

The SQL interval types are mapped to the XML Schema types **`xs:yearMonthDuration`** and **`xs:dayTimeDuration`**. The **`xs:pattern`** facet is used to require precisely the year, month, day, hour, minute and second fields indicated by the SQL type. The **`xs:pattern`** facet 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 domain is mapped to XML by mapping the domain's data type to XML and then optionally applying to the generated XML Schema type an annotation that identifies the name of the domain.

An SQL distinct type is mapped to an XML Schema simple type by mapping the source type of the distinct type. Optionally, an annotation specifying the name of the distinct type is applied to the generated XML Schema 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.

4.10.6 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 9.8, "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"**.

4.10.7 Mapping XQuery atomic values to SQL values

As defined in [XQueryDM], 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 **xs:yearMonthDuration**, or derived from **xs:yearMonthDuration**, then **xs:yearMonthDuration**.

— If ***AT*** is **xs:dayTimeDuration**, or derived from **xs:dayTimeDuration**, then **xs:dayTimeDuration**.

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

This part of ISO/IEC 9075 (notably, in Subclause 6.6, "<XML 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 ***XDNV*** of ***AV*** is positive, and the XQuery datetime timezone component of ***AV*** is an empty XQuery sequence, then ***XDNV*** is regarded as being a value of type TIMESTAMP WITHOUT TIME ZONE. If ***XDNV*** would have a SECOND field greater than 59 in a minute of UTC that has exactly 59 seconds, then it is implementation-defined whether an implementation-defined value of type TIMESTAMP WITHOUT TIME ZONE is identified with ***XDNV***, or whether an exception condition is raised: *data exception — datetime field overflow*.

— 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 AV denotes a minute of UTC that has exactly 59 seconds and the SECOND field of ***AV*** is greater than 59, then it is implementation-defined whether an implementation-defined value of type TIMESTAMP WITHOUT TIME ZONE is identified with ***AV***, or whether an exception condition is raised: *data exception — datetime field overflow*.

— 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.

4.10.8 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, XML(SEQUENCE), or a type defined in some part of ISO/IEC 9075 other than [ISO9075-2] and this part. 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 6.5, "<cast specification>". 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*.

4.10.9 Mapping an SQL table to XML

Subclause 9.11, "Mapping an SQL table to XML and an XML Schema document", defines a mapping of an SQL table to one or both of two documents: an XML Schema document that describes the structure of the mapped XML and either an XML document or a sequence of XML elements. Only base tables and viewed tables may be the source of this mapping.

NOTE 15 — This part of this International Standard specifies no syntax for invoking the mapping specified in Subclause 9.11, "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. It is the responsibility of any such application or other standard to ensure that the correct number of arguments as well as a valid value for each argument are supplied for this mapping.

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 a sequence of XML 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"`**.

— Whether to map the table into XML data, to map the table into an XML Schema document, or both.

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

The rules of Subclause 9.11, "Mapping an SQL table to XML and an XML Schema document", are supported by the rules of Subclause 9.9, "Mapping an SQL table to XML Schema data types", and Subclause 9.10, "Mapping an SQL table to an XML element or a sequence of XML elements".

4.10.10 Mapping an SQL schema to XML

Subclause 9.14, "Mapping an SQL schema to an XML document and an XML Schema document", defines a mapping between the tables of an SQL-schema and either or both of two documents: an XML document that represents the data in these tables, and an XML Schema document that describes the first document.

NOTE 16 — This part of this International Standard specifies no syntax for invoking the mapping specified in Subclause 9.14, "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. It is the responsibility of any such application or other standard to ensure that the correct number of arguments as well as a valid value for each argument are supplied for this mapping.

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 mapping allows the invoker to specify:

— Whether to map each table to a sequence of XML 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 each table to a single XML 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"`**.

— Whether to map the schema into XML data, to map the schema into an XML Schema document, or both.

Some of the XML Schema type definitions and element declarations 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.

The rules of Subclause 9.14, "Mapping an SQL schema to an XML document and an XML Schema document", are supported by the rules of Subclause 9.12, "Mapping an SQL schema to XML Schema data types", and Subclause 9.13, "Mapping an SQL schema to an XML element".

4.10.11 Mapping an SQL catalog to XML

Subclause 9.17, "Mapping an SQL catalog to an XML document and an XML Schema document", defines a mapping between the tables of an SQL catalog and either or both of two documents: an XML document that represents the data in these tables, and an XML Schema document that describes the first document.

NOTE 17 — This part of this International Standard specifies no syntax for invoking the mapping specified in Subclause 9.17, "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. It is the responsibility of any such application or other standard to ensure that the correct number of arguments as well as a valid value for each argument are supplied for this mapping.

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 each table to a sequence of XML 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 each table to a single XML 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"**.

— Whether to map the catalog into XML data, to map the catalog into an XML Schema document, or both.

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

The rules of Subclause 9.17, "Mapping an SQL catalog to an XML document and an XML Schema document", are supported by the rules of Subclause 9.15, "Mapping an SQL catalog to XML Schema data types", and Subclause 9.16, "Mapping an SQL catalog to an XML element".

(Blank page)

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 | BOM
  | COLUMNS | CONTENT
  | DOCUMENT
  | ENCODING
  | HEX
  | ID | INDENT
  | LOCATION
  | NAMESPACE | NIL
  | PATH | PRESERVE
  | SEQUENCE | STANDALONE | STRIP
  | UNTYPED | URI
  | VALID | VERSION
  | WHITESPACE
  | XMLSCHEMA | XMLDECLARATION
<reserved word> ::=
```

```
    !! All alternatives from ISO/IEC 9075-2

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

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.

(Blank page)

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> ::=
  <primary XML type modifier>
      [ <left paren> <secondary XML type modifier> <right paren> ]

<primary XML type modifier> ::=
    DOCUMENT
  | CONTENT
  | SEQUENCE

<secondary XML type modifier> ::=
    ANY
  | UNTYPED
  | XMLSCHEMA <XML valid according to what> [ <XML valid element clause> ]
```

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 SEQUENCE, CONTENT(ANY), or CONTENT(UNTYPED) is implicit.

3) Insert this SR Case:

   a) If <primary XML type modifier> specifies SEQUENCE, then <secondary XML type modifier> shall not be specified.

b) Otherwise, if <secondary XML type modifier> is not specified, then it is implementation-defined whether UNTYPED or ANY is implicit.

4) Insert this SR If <secondary XML type modifier> specifies XMLSCHEMA, then let *RXS* be the indicated registered XML Schema, let *ENSURI* be the indicated XML namespace, if any, and let ***GEDSC*** be the indicated global element declaration schema component, if any.

NOTE 18 — Indicated registered XML Schema, indicated XML namespace, and indicated global element declaration schema component are defined in Subclause 11.6, "<XML valid according to clause>".

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 21) If <data type> is an <XML type>, then an XML type descriptor is created, including the following:

a) The name of the data type (XML).

b) The primary XML type modifier described by the <primary XML type modifier> (DOCUMENT, CONTENT, or SEQUENCE).

c) The secondary XML type modifier described by the <secondary XML type modifier> (UNTYPED, ANY, or XMLSCHEMA), if any.

d) The registered XML Schema descriptor of the indicated registered XML Schema, if any.

e) The XML namespace URI of the indicated XML namespace, if any.

f) The XML NCName of the indicated global element declaration schema component, if any.

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(DOCUMENT(UNTYPED)) type", conforming SQL language shall not contain an <XML type> whose <primary XML type modifier> is DOCUMENT and <secondary XML type modifier> is UNTYPED.

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

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

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

8) Insert this CR Without Feature X191, “XML(DOCUMENT(XMLSCHEMA)) type”, conforming SQL language shall not contain an <XML type> whose <primary XML type modifier> is DOCUMENT and <secondary XML type modifier> specifies XMLSCHEMA.

9) Insert this CR Without Feature X192, “XML(CONTENT(XMLSCHEMA)) type”, conforming SQL language shall not contain an <XML type> whose <primary XML type modifier> is CONTENT and <secondary XML type modifier> specifies XMLSCHEMA.

10) Insert this CR Without Feature X260, “XML type: ELEMENT clause”, conforming SQL language shall not contain an <XML type> that contains <XML valid element clause>.

11) Insert this CR Without Feature X261, “XML type: NAMESPACE without ELEMENT clause”, conforming SQL language shall not contain an <XML type> that contains an <XML valid element clause> that does not contain an <XML valid element name specification>.

12) Insert this CR Without Feature X263, “XML type: NO NAMESPACE with ELEMENT clause”, conforming SQL language shall not contain an <XML type> that contains an <XML valid element namespace specification> that contains NO NAMESPACE.

13) Insert this CR Without Feature X264, “XML type: schema location”, conforming SQL language shall not contain an <XML type> that contains <XML valid schema location>.

14) 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 <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>.

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>, <next value expression>, or <routine invocation>.

Conformance Rules

No additional Conformance Rules.

6.4 <case expression>

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

Function

Specify a conditional value.

Format

```
<when operand> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <XML content predicate part 2>
  | <XML document predicate part 2>
  | <XML valid predicate part 2>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

6.5 <cast specification>

This Subclause modifies Subclause 6.13, "<cast specification>", in ISO/IEC 9075-2.

Function

Specify a data conversion.

Format

```
<cast specification> ::=
  CAST <left paren> <cast operand> AS <cast target>
      [ FORMAT <cast template> ]
      [ <XML passing mechanism> ]
      <right paren>
```

Syntax Rules

1) Insert after SR 3) If <XML passing mechanism> is specified, then <cast operand> shall be a <value expression> and both *SD* and *TD* shall be XML types.

2) Insert after SR 3) If *SD* and *TD* are both XML types and <XML passing mechanism> is not specified, then it is implementation-defined whether BY REF or BY VALUE is implicit.

3) Insert after SR 7) Add a new rightmost column to the table in 7th paragraph:

SD	*TD*
	XML
EN	N
AN	N
C	N
D	N
T	N
TS	N
YM	N
DT	N
BO	N
UDT	N
B	N
RT	N
CT	N
RW	N

4) Insert after SR 7) Add new rows at the end of the table in 7th paragraph:

```
     XML     N   N   N   N   N   N   N   N   N   N   N   N   N   N   M
Where:
     XML  = XML type
```

5) Insert before SR 16) If BY REF is specified or implied, then

Case:

a) If *TD* is either XML(DOCUMENT(UNTYPED)) or XML(CONTENT(UNTYPED)), then *SD* shall be either XML(DOCUMENT(UNTYPED)) or XML(CONTENT(UNTYPED)).

b) If *TD* is either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)), then

Case:

i) If the type descriptor of *TD* includes an XML namespace URI ***N*** and an XML NCName ***EN*** of a global element declaration schema component, then *SD* shall be either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) such that the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *TD* is identical to the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *SD* and the type descriptor of *SD* includes an XML namespace URI that is identical to ***N***, as defined by [Namespaces], and an XML NCName that is equivalent to ***EN***.

ii) If the type descriptor of *TD* includes an XML namespace URI ***N***, then *SD* shall be either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) such that the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *TD* is identical to the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *SD* and the type descriptor of *SD* includes an XML namespace URI that is identical to ***N***, as defined by [Namespaces].

iii) Otherwise, *SD* shall be either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) such that the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *TD* is identical to the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *SD*.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 3) If *SD* and *TD* are both XML types, then:

a) Case:

i) If *TD* is XML(DOCUMENT(UNTYPED)), XML(DOCUMENT(ANY)), or XML(DOCUMENT(XMLSCHEMA)), and *SV* is not an XQuery sequence of length 1 (one) whose sole XQuery item is an XQuery document node whose **children** property contains exactly one XQuery element node, zero or more XQuery comment nodes, zero or more XQuery processing instruction nodes, then an exception condition is raised: *data exception — not an XML document*.

ii) If *TD* is XML(CONTENT(UNTYPED)), XML(CONTENT(ANY)), or XML(CONTENT(XMLSCHEMA)), and *SV* 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) Case:

i) If BY VALUE is specified or implied, then

Case:

1) If *TD* is either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)), then let *S* be the <secondary XML type modifier> contained in <cast target>.

   A) Case:

      I) If *TD* is XML(DOCUMENT(XMLSCHEMA)), then let *DCS* be DOCUMENT.

      II) Otherwise, let *DCS* be CONTENT.

   B) *TV* is the result of

      ```
      XMLVALIDATE( DCS VE ACCORDING TO S )
      ```

2) If *TD* is either XML(DOCUMENT(ANY)), XML(CONTENT(ANY)), or XML(SEQUENCE), then the General Rules of Subclause 10.18, "Constructing a copy of an XML value", are applied with ***SV*** as *VALUE*; let ***TV*** be the *COPY* returned from the application of those General Rules.

3) Otherwise, the General Rules of Subclause 10.19, "Constructing an unvalidated XQuery document node", are applied with ***SV*** as *XQUERYDOCNODE*; let ***TV*** be the *UNVALIDATEDDOCNODE* returned from the application of those General Rules.

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

Conformance Rules

No additional Conformance Rules.

6.6 <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> [ <XML passing mechanism> ] <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* 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) 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>. <data type> shall not contain a <collate clause>.

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 distinct type whose source type is 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(DOCUMENT(UNTYPED)), XML(DOCUMENT(ANY)), XML(DOCUMENT(XMLSCHEMA)), or XML(CONTENT(XMLSCHEMA)), 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 *VE*.

9) If <XML passing mechanism> is specified, then:

   a) <XML cast operand> shall be a <value expression>.

   b) *SD* and *TD* shall both be XML types.

10) If the <XML cast operand> is a <value expression> *VE*, then

    Case:

    a) If *VE* simply contains a <dynamic parameter specification> *DPS*, then the <XML cast specification> is equivalent to

       ```
       CAST ( DPS AS TD )
       ```

    b) Otherwise:

       i) *SD* shall not be a collection type, a distinct type whose source type is a collection type, a row type, a structured type or a reference type.

       ii) If *SD* and *TD* are both XML types, then

           1) If <XML passing mechanism> is not specified, then it is implementation-defined whether BY REF or BY VALUE is implicit.

           2) Let *XPM* be the implicit or explicit <XML passing mechanism>.

           3) The <XML cast specification> is equivalent to

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

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

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

12) If *TD* is character string type, then 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*.

13) 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*.

14) Case:

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

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

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

15) Case:

a) If *SQLT* is a character string type, then let *XT* be **xs:string**.

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

Case:

i) If *XBE* specifies BASE64, then let *XT* be **xs:base64Binary**.

ii) Otherwise, let *XT* be **xs:hexBinary**.

c) If *SQLT* is an exact numeric type, then

Case:

i) If the type designator of *SQLT* is DECIMAL or NUMERIC, then let *XT* be **xs:decimal**.

ii) If the type designator of *SQLT* is SMALLINT, INTEGER, or BIGINT, then let *XT* be **xs:integer**.

d) If *SQLT* is an approximate numeric type, then let *XT* be **xs:double**.

e) If *SQLT* is a datetime type, then

Case:

i) If the type designator of *SQLT* is DATE, then let *XT* be **xs:date**.

ii) If the type designator of *SQLT* is TIME WITH TIME ZONE, then let *XT* be **xs:time**.

iii) If the type designator of *SQLT* is TIME WITHOUT TIME ZONE, then let *XT* be **xs:time**.

iv) If the type designator of *SQLT* is TIMESTAMP WITH TIME ZONE, then let *XT* be **xs:dateTime**.

v) If the type designator of *SQLT* is TIMESTAMP WITHOUT TIME ZONE, then let *XT* be **xs:dateTime**.

f) Otherwise, the General Rules of Subclause 9.5, "Mapping SQL data types to XML Schema data types", are applied with *SQLT* as *SQLTYPE*, *ENC* as *ENCODING*, and "absent" as *NULLS*; let *XT* be the *SCHEMA TYPE* returned from the application of those General Rules.

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 6.5, "<cast specification>", are applied to

```
CAST ( AVE AS TD )
```

2) If *SD* is a distinct type, then let *SSD* be the source type of *SD*. The Access Rules of Subclause 6.5, "<cast specification>", are applied to

```
CAST ( VE 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, in an <SQL schema statement>, 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 applied.

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

   a) The General Rules of Subclause 9.8, "Mapping values of SQL data types to values of XML Schema data types", are applied with *V* as *SQLVALUE*, *ENC* as *ENCODING*, "absent" as *NULLS*, and *True* as *CHARMAPPING*; let *XMLV* be the *XMLVALUE* returned from the application of those General Rules.

   b) Let *TEMPV* be result of

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

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

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

      i) The General Rules of Subclause 9.5, "Mapping SQL data types to XML Schema data types", are applied with *SD* as *SQLTYPE*, *ENC* as *ENCODING*, and "absent" as *NULLS*; let ***XST*** be the *SCHEMA TYPE* returned from the application of those General Rules.

      ii) Case:

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

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

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

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

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

      iv) The Syntax Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied with the null value as *NONTERMINAL*; let ***XSC*** be the *STATICCONTEXT* returned from the application of those Syntax Rules. The General Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied; let ***XDC*** be the *DYNAMICCONTEXT* returned from the application of those General Rules.

NOTE 19 — ***XSC*** is an XQuery static context. ***XDC*** is an XQuery dynamic context.

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

vi) Case:

1) If the SQL-implementation supports Feature X211, "XML 1.1 support", then 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*.

2) Otherwise, let *TV* be the result of the XQuery evaluation with XML 1.0 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) The result of the <XML cast specification> is *TV*.

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

a) The General Rules of Subclause 10.17, "Removing XQuery document nodes from an XQuery sequence", are applied with ***V*** as *SEQUENCE*; let ***XV*** be the *RESULT* returned from the application of those General Rules.

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

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

d) Let ***XMLT*** be an XML 1.1 QName for ***XT***.

NOTE 20 — ***XMLT*** may be in one of the built-in namespaces denoted by the prefix **xs**, or it may be in an implementation-dependent namespace, not necessarily available to the user.

e) The Syntax Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied with the null value as *NONTERMINAL*; let ***XSC*** be the *STATICCONTEXT* returned from the application of those Syntax Rules.

NOTE 21 — ***XSC*** is an XQuery static context.

f) The General Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied; let ***XDC*** be the *DYNAMICCONTEXT* returned from the application of those General Rules.

NOTE 22 — ***XDC*** is an XQuery dynamic context.

g) Let ***XSC*** and ***XDC*** be augmented with an XQuery variable **$TEMP** whose XQuery formal type notation is "**xs:anyAtomicType**" and whose value is ***AV***.

h) 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

Case:

i) If *SQLT* is TIMESTAMP WITHOUT TIME ZONE, then

```
fn:adjust-dateTime-to-timezone(
  fn:adjust-dateTime-to-timezone( $TEMP cast as xs:dateTime,
                                  xs:dayTimeDuration("PT0H")
                                ), () ) cast as XMLT
```

ii) If *SQLT* is TIME WITHOUT TIME ZONE, then

```
fn:adjust-time-to-timezone(
  fn:adjust-time-to-timezone( $TEMP cast as xs:time,
                              xs:dayTimeDuration("PT0H")
                            ), () ) cast as XMLT
```

iii) If *SQLT* is DATE, then

```
fn:adjust-date-to-timezone(
  fn:adjust-date-to-timezone( $TEMP cast as xs:date,
                              xs:dayTimeDuration("PT0H")
                            ), () ) cast as XMLT
```

iv) Otherwise,

```
$TEMP cast as XMLT
```

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

i) ***BV*** is an XQuery atomic value.

NOTE 23 — The following rules are based on the fact 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 4.10.7, "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 derived from **xs:hexBinary**, then let *A* be an arbitrary <literal> of binary string type whose value is ***BV*** decoded as hex.

iii) If ***XT*** is **xs:base64Binary** or is derived from **xs:base64Binary**, then let *A* be an arbitrary <literal> of binary string type whose value is ***BV*** decoded as base64.

iv) 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***.

v) 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***.

vi) If ***XT*** is **xs:date** or derived from **xs:date**, 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) Let *A* be an arbitrary <literal> of declared type *SQLT* whose value is ***BV*** .

vii) 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 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

Case:

A) If the XQuery datetime timezone component of ***AV*** is the empty XQuery sequence, then:

I) The Syntax Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied with the null value as *NONTERMINAL*; let ***XSC2*** be the *STATICCONTEXT* returned from the application of those Syntax Rules.

NOTE 24 — ***XSC2*** is an XQuery static context.

II) The General Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied; let ***XDC2*** be the *DYNAMICCONTEXT* returned from the application of those General Rules.

NOTE 25 — ***XDC2*** is an XQuery dynamic context.

III) Let *XSC2* and *XDC2* be augmented with an XQuery variable $TEMP whose XQuery formal type notation is "**xs:anyAtomicType**" and whose value is ***BV***.

IV) Let ***CV*** be the result of the XQuery evaluation with XML 1.1 lexical rules, using *XSC2* and *XDC2* as the XQuery expression context, of the XQuery expression

```
fn:adjust-dateTime-to-timezone(
  $TEMP cast as xs:dateTime,
  xs:dayTimeDuration("PT0H") ) cast as XMLT
```

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

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

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

Case:

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

I) The Syntax Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied with the null value as *NONTERMINAL*; let ***XSC3*** be the *STATICCONTEXT* returned from the application of those Syntax Rules.

NOTE 26 — ***XSC3*** is an XQuery static context.

II) The General Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied; let ***XDC3*** be the *DYNAMICCONTEXT* returned from the application of those General Rules.

NOTE 27 — ***XDC3*** is an XQuery dynamic context.

III) Let *XSC3* and *XDC3* be augmented with an XQuery variable $TEMP whose XQuery formal type notation is "**xs:anyAtomicType**" and whose value is ***BV***.

IV) Let ***CV*** be the result of the XQuery evaluation with XML 1.1 lexical rules, using *XSC3* and *XDC3* as the XQuery expression context, of the XQuery expression

```
fn:adjust-time-to-timezone(
  $TEMP cast as xs:time,
  xs:dayTimeDuration("PT0H") ) cast as XMLT
```

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

B) Otherwise, 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***).

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

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

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

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

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

NOTE 28 — This may raise a warning or 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

1) Without Feature X025, "XMLCast", conforming SQL language shall not contain an <XML cast specification>.

6.7 <value expression>

This Subclause modifies Subclause 6.28, "<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)c)ii) A <cast specification> whose result type is an XML type and whose <cast operand> has a declared type that is an XML type and whose <XML passing mechanism> specifies or implies BY VALUE.

3) Insert after SR 7)o) An <aggregate function> that specifies <XML aggregate>.

4) Insert after SR 7)y) 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 non-deterministic registered XML Schema and *XVP* does not specify an <XML valid element name specification> of an <XML valid element namespace specification>.

   c) *XVP* specifies an <XML valid according to clause> that identifies a non-deterministic XML namespace and *XVP* does not specify an <XML valid element name specification>.

   d) *XVP* specifies an <XML valid element clause> that identifies a non-deterministic global element declaration schema component of a registered XML Schema.

   NOTE 29 — 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 namespace specification> that identifies a deterministic XML namespace of a registered XML Schema, or contains an <XML valid element name specification> that identifies a deterministic global element declaration schema component of a registered XML Schema.

5) Insert after SR 7)y) An <XML value function> other than <XML query> and <XML concatenation>.

6) Insert after SR 7)y) An <XML query> that does not conform to implementation-defined rules enabling the SQL-implementation to deduce that the result of the <XML query> is deterministic.

7) Insert after SR 7)y) An <XML concatenation> that does not implicitly or explicitly specify RETURNING SEQUENCE.

8) Insert after SR 7)y) An <XML character string serialization> or an <XML binary string serialization>.

9) Insert after SR 7)y) An <XML exists predicate> that does not conform to implementation-defined rules enabling the SQL-implementation to deduce that the result of the <XML exists predicate> is deterministic.

10) Insert after SR 7)y) An <XML cast specification> whose <XML cast target> is an XML type, whose <XML cast operand> has a declared type that is an XML type, and whose <XML passing mechanism> specifies or implies BY VALUE.

11) Insert after SR 7)y) An <XML cast specification> whose <XML cast target> is either XML(CONTENT(ANY)) or XML(CONTENT(UNTYPED)) type and whose <XML cast operand> has a declared type that is not an XML type.

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.

6.8 <string value function>

This Subclause modifies Subclause 6.32, "<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 from ISO/IEC 9075-2
  | <XML character string serialization>

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

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

<XML serialize bom> ::=
  WITH [ NO ] BOM

<XML declaration option> ::=
    INCLUDING XMLDECLARATION
  | EXCLUDING XMLDECLARATION

<XML serialize indent> ::=
  [ NO ] INDENT

<document or content> ::=
    DOCUMENT
  | CONTENT

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

<XML binary string serialization> ::=
  XMLSERIALIZE <left paren> [ <document or content> ]
      <XML value expression> AS <data type>
      [ ENCODING <XML encoding specification> ]
      [ <XML serialize bom> ]
      [ <XML serialize version> ]
      [ <XML declaration option> ]
      [ <XML serialize indent> ]
      <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>, <regex substring function>, <fold>, <transcoding>, <character transliteration>, <regex 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) If <document or content> is not specified, then a <document or content> that specifies CONTENT is implicit.

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

   c) 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].

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

3) Replace SR 17) The declared type of <binary value function> is the declared type of the immediately contained <binary substring function>, <binary trim function>, <binary overlay function>, or <XML binary string serialization>.

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

   a) If <document or content> is not specified, then a <document or content> that specifies CONTENT is implicit.

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

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

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

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

   f) 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].

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

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>, <regex substring function>, <fold>, <transcoding>, <character transliteration>, <regex 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

   a) Let *DCS* be the explicit or implicit <document or content>.

   b) Let *XMLV* be the value of the <XML value expression>.

   c) Let *DT* be the <data type>.

   d) Let *CS* be the character set of *DT*.

   e) Case:

      i) If <XML serialize bom> is specified, then

         Case:

         1) If WITH BOM is specified, then let *B* be *True*.

         2) Otherwise, let *B* be *False*.

      ii) Otherwise, let *B* be *Unknown*.

   f) Let *VER* be the <string value expression> simply contained in the explicit or implicit <XML serialize version>.

   g) Case:

      i) If <XML declaration option> is specified and specifies INCLUDING XMLDECLARATION, then let *DECL* be *True*.

      ii) If <XML declaration option> is specified and specifies EXCLUDING XMLDECLARATION, then let *DECL* be *False*.

      iii) Otherwise, let *DECL* be *Unknown*.

   h) Case:

      i) If <XML serialize indent> is specified and does not contain NO, then let *IND* be *True*.

      ii) Otherwise, let *IND* be *False*.

   i) The General Rules of Subclause 10.15, "Serialization of an XML value", are applied with *XMLV* as *VALUE*, *DCS* as *SYNTAX*, *DT* as *TYPE*, *CS* as *ENCODING*, *B* as *BOMDIA*, *VER* as *VERSION*, *DECL* as *XMLDECLARATION*, and *IND* as *INDENT*; let result of <XML character string serialization> be the *SERIALIZATION* returned from the application of those General Rules

3) Replace GR 13) The result of <binary value function> is the result of the simply contained <binary substring function>, <binary trim function>, <binary overlay function>, or <XML binary string serialization>.

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

   a) Let *DCS* be the explicit or implicit <document or content>.

b) Let *XMLV* be the value of the <XML value expression>.

c) Let *DT* be the <data type>.

d) Let *XEN* be the <XML encoding name>.

e) Case:

i) If <XML serialize bom> is specified, then

Case:

1) If WITH BOM is specified, then let *B* be *True*.

2) Otherwise, let *B* be *False*.

ii) Otherwise, let *B* be *Unknown*.

f) Let *VER* be the <string value expression> simply contained in the explicit or implicit <XML serialize version>.

g) Case:

i) If <XML declaration option> is specified and specifies INCLUDING XMLDECLARATION, then let *DECL* be *True*.

ii) If <XML declaration option> is specified and specifies EXCLUDING XMLDECLARATION, then let *DECL* be *False*.

iii) Otherwise, let *DECL* be *Unknown*.

h) Case:

i) If <XML serialize indent> is specified and does not contain NO, then let *IND* be *True*.

ii) Otherwise, let *IND* be *False*.

i) The General Rules of Subclause 10.15, "Serialization of an XML value", are applied with *XMLV* as *VALUE*, *DCS* as *SYNTAX*, *DT* as *TYPE*, *XEN* as *ENCODING*, *B* as *BOMDIA*, *VER* as *VERSION*, *DECL* as *XMLDECLARATION*, and *IND* as *INDENT*; let result of <XML binary string serialization> be the *SERIALIZATION* returned from the application of those General Rules

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> 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> that is DOCUMENT.

3) Insert this CR Without Feature X072, "XMLSerialize: character string serialization", conforming SQL language shall not contain an <XML character string serialization>.

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

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

6) Insert this CR Without Feature X075, “XMLSerialize: binary string serialization”, conforming SQL language shall not contain an <XML binary string serialization>.

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.

9) Insert this CR Without Feature X077, “XMLSerialize: explicit ENCODING option”, conforming SQL language shall not contain an <XML binary string serialization> that contains ENCODING.

10) Insert this CR Without Feature X078, “XMLSerialize: explicit XML declaration”, in conforming SQL language, <XML character string serialization> shall not contain XMLDECLARATION.

11) Insert this CR Without Feature X078, “XMLSerialize: explicit XML declaration”, in conforming SQL language, <XML binary string serialization> shall not contain XMLDECLARATION.

12) Insert this CR Without Feature X068, “XMLSerialize: BOM”, in conforming SQL language, <XML serialize bom> shall not be specified.

13) Insert this CR Without Feature X069, “XMLSerialize: INDENT”, in conforming SQL language, <XML serialize indent> shall not be specified.

6.9 <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 the <value expression primary> immediately contained in <XML 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>.

6.10 <XML value function>

Function

Specify a function that yields a value of type XML.

Format

```
<XML value function> ::=
    <XML comment>
  | <XML concatenation>
  | <XML document>
  | <XML element>
  | <XML forest>
  | <XML parse>
  | <XML PI>
  | <XML query>
  | <XML text>
  | <XML validate>
```

Syntax Rules

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

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 document>, <XML element>, <XML forest>, <XML parse>, <XML PI>, <XML query>, <XML text>, or <XML validate>.

Conformance Rules

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

6.11 <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> <character value expression>
      [ <XML returning clause> ] <right paren>
```

Syntax Rules

1) If <XML returning clause> is not specified, then it is implementation-defined whether RETURNING CONTENT or RETURNING SEQUENCE is implicit.

2) Let *SVE* be the <character value expression>.

3) Case:

   a) If RETURNING CONTENT is specified or implied, then <XML comment> is equivalent to

      ```
      XMLDOCUMENT ( XMLCOMMENT ( SVE RETURNING SEQUENCE ) RETURNING CONTENT )
      ```

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

   NOTE 30 — If RETURNING CONTENT is specified or implied, then the declared type of <XML comment> is effectively established by the Syntax Rules of Subclause 6.13, "<XML document>".

Access Rules

None.

General Rules

1) Let *SV* be the value of *SVE*.

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

3) If the value of

   ```
   '<!--' || SV || '-->'
   ```

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

4) Let ***XCII*** be an XQuery comment node with the following properties:

a) The value of the **content** property is the value of *SV*, converted to Unicode using the implementation-defined mapping from the character set of *SV* to Unicode.

b) The value of the **parent** property is set to **empty**.

5) 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, "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.

6.12 <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 it is implementation-defined whether RETURNING CONTENT or RETURNING SEQUENCE is implicit.

2) Let *N* be the number of <XML value expression>s simply contained in <XML concatenation>. Let XVE_i, 1 (one) ≤ *i* ≤ *N*, be the *i*-th <XML value expression> and let T_i be the declared type of XVE_i.

3) Case:

   a) If RETURNING CONTENT is specified or implied, then the declared type of <XML concatenation> is

      Case:

      i) If, for all *i*, 1 (one) ≤ *i* ≤ *N*, T_i is either XML(DOCUMENT(UNTYPED)) or XML(CONTENT(UNTYPED)), then XML(CONTENT(UNTYPED)).

      ii) If there exists a registered XML Schema *S* and a global element declaration schema component ***E*** such that, for all *i*, 1 (one) ≤ *i* < *N*, T_i is XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) and the XML type descriptor of T_i includes the registered XML Schema descriptor of *S* and an XML namespace URI ***NS*** that is identical to the XML Namespace URI of ***E***, as defined by [Namespaces], and an XML NCName ***EN*** that is equivalent to the XML NCName of *E*, then XML(CONTENT(XMLSCHEMA)), whose XML type descriptor includes the registered XML Schema descriptor of *S* and the XML namespace URI ***NS*** and the XML NCName ***EN***.

      iii) If there exists a registered XML Schema *S* and an XML namespace URI ***NS*** such that, for all *i*, 1 (one) ≤ *i* < *N*, T_i is XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) and the XML type descriptor of T_i includes the registered XML Schema descriptor of *S* and an XML namespace URI that is identical to ***NS***, as defined by [Namespaces], then XML(CONTENT(XMLSCHEMA)), whose XML type descriptor includes the registered XML Schema descriptor of *S* and the XML namespace URI ***NS***.

      iv) If there exists a registered XML Schema *S* such that, for all *i*, 1 (one) ≤ *i* < *N*, T_i is XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) and the XML type descriptor of T_i includes the registered XML Schema descriptor of *S*, then XML(CON-

TENT(XMLSCHEMA)), whose XML type descriptor includes the registered XML Schema descriptor of *S*.

v) Otherwise, XML(CONTENT(ANY)).

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

4) If RETURNING CONTENT is specified or implied, then

a) Let *RT* be the declared type of <XML concatenation>.

b) <XML concatenation> is equivalent to

```
CAST(
  XMLDOCUMENT (
    XMLCONCAT ( XVE_1, XVE_2, ..., XVE_N RETURNING SEQUENCE )
    RETURNING CONTENT )
  AS RT )
```

Access Rules

None.

General Rules

1) For all i, 1 (one) ≤ i ≤ N, let XV_i be the results of XVE_i.

2) Let R_0 be the null value.

3) For all i, 1 (one) ≤ i ≤ N, the General Rules of Subclause 10.14, "Concatenation of two XML values", are applied with R_{i-1} as *FIRSTVAL* and XV_i as *SECONDVAL*; let R_i be the *CONCAT* returned from the application of those General Rules.

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

Conformance Rules

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

2) Without Feature X241, "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.

6.13 <XML document>

Function

Generate an XML value with a single XQuery document node.

Format

```
<XML document> ::=
  XMLDOCUMENT <left paren> <XML value expression>
      [ <XML returning clause> ] <right paren>
```

Syntax Rules

1) If <XML returning clause> is not specified, then it is implementation-defined whether RETURNING CONTENT or RETURNING SEQUENCE is implicit.

2) Let *XVE* be the <XML value expression> simply contained in the <XML document>.

3) Case:

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

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

Access Rules

None.

General Rules

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

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

3) The Syntax Rules of Subclause 10.21, "Determination of an XQuery formal type notation", are applied with *XV* as *SOURCE* and *False* as *CONTEXT*; let ***XVFT*** be the *FORMALTYPE* returned from the application of those Syntax Rules.

4) The Syntax Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied with the null value as *NONTERMINAL*; let ***XSC*** be the *STATICCONTEXT* returned from the application of those Syntax Rules.

   NOTE 31 — ***XSC*** is an XQuery static context.

5) The **`construction mode`** component of ***XSC*** is set to **preserve**.

6) The General Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied; let ***XDC*** be the *DYNAMICCONTEXT* returned from the application of those General Rules.

NOTE 32 — ***XDC*** is an XQuery dynamic context.

7) Let ***XSC*** and ***XDC*** be augmented with an XQuery variable **$EXPR**, whose XQuery formal type notation is ***XVFT***, and whose value is *XV*.

8) Case:

   a) If Feature X211, "XML 1.1 support" is supported, then let *DN* 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

      ```
      document { $EXPR }
      ```

      If an XQuery error occurs during the evaluation, then an exception condition is raised: *XQuery error*.

   b) Otherwise, let *DN* be the result of an XQuery evaluation with XML 1.0 lexical rules, using ***XSC*** and ***XDC*** as the XQuery expression context of the XQuery expression

      ```
      document { $EXPR }
      ```

      If an XQuery error occurs during the evaluation, then an exception condition is raised: *XQuery error*.

9) Case:

   a) If the declared type of the <XML document> is XML(CONTENT(UNTYPED)), then the General Rules of Subclause 10.19, "Constructing an unvalidated XQuery document node", are applied with ***DN*** as *XQUERYDOCNODE*; let result of the <XML document> be the *UNVALIDATEDDOCNODE* returned from the application of those General Rules.

   b) Otherwise, the result of the <XML document> is *DN*.

Conformance Rules

1) Without Feature X030, "XMLDocument", conforming SQL language shall not contain an <XML document>.

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

6.14 <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) For *i* ranging 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 either a predefined type other than XML or a distinct type whose source type is neither an XML type nor a collection 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. The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with CN_i as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***ANC***$_i$ be the *XMLNAME* returned from the application of those General Rules.

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 10.12, "Determination of namespace URI", are applied with ***ANC***$_i$ as *QNAME* and the <XML element> as *BNFTERM*; let ***NSURI***$_i$ be the *NSURI* returned from the application of those Syntax Rules.

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 10.12, "Determination of namespace URI", are applied with ***EN*** as *QNAME* and the <XML element> as *BNFTERM*; let ***ENSURI*** be the *NSURI* returned from the application of those Syntax Rules.

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 data type, or a distinct type.

8) Case:

a) If <XML element content> is specified, then:

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

ii) Let *OPT* be the explicit or implicit <XML content option>.

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

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 it is implementation-defined whether RETURNING CONTENT or RETURNING SEQUENCE is implicit.

11) Let *XEN* be the <XML element name>.

12) If <XML namespace declaration> is specified, then let *XND* be "<comma> <XML namespace declaration>"; otherwise, let *XND* be a zero-length string.

13) If <XML attributes> is specified, then let *XAT* be "<comma> <XML attributes>"; otherwise, let *XAT* be the zero-length string.

14) Let *k* be the number of <XML element content>s immediately contained in <XML element>. For each such <XML element content> in order of its position in <XML element> from left to right, let $XMLEC_l$, 1 (one) ≤ *l* ≤ *k*, be "<comma> <XML element content>".

15) Let *XCO* be the specified or implied <XML content option>.

16) Case:

    a) If RETURNING CONTENT is specified or implied, then <XML element> is equivalent to

```
XMLDOCUMENT (
  XMLELEMENT (
    NAME XEN XND XAT XMLEC1 XMLEC2 ... XMLECk XCO
    RETURNING SEQUENCE )
  RETURNING CONTENT )
```

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

NOTE 33 — If RETURNING CONTENT is specified or implied, then the declared type of <XML element> is effectively established by the Syntax Rules of Subclause 6.13, "<XML document>".

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 6.5, "<cast specification>", 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) For *i* range from 1 (one) to *n*, if the value of AV_i is not the null value, then:

    a) The General Rules of Subclause 9.8, "Mapping values of SQL data types to values of XML Schema data types", are applied with AV_i as *SQLVALUE*, *ENC* as *ENCODING*, "absent" as *NULLS*, and *False*

as *CHARMAPPING*; let CAV_i be the *XMLVALUE* returned from the application of those General Rules. $\mathbf{CAV}_i$ is a character string of Unicode characters.

b) Let $\mathbf{AII}_i$ 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 $\mathbf{ANC}_i$ and whose namespace URI is $\mathbf{NSURI}_i$.

ii) The **string-value** property is $\mathbf{CAV}_i$.

iii) The **type-name** property is `xs:untypedAtomic`.

iv) The **parent** property is initially unknown.

4) Let **ATTRS** be the XQuery sequence of XQuery attribute nodes $\mathbf{AII}_i$, 1 (one) ≤ i ≤ n, such that AV_i is not the null value.

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 $V_1, \ldots, V_N$.

6) The General Rules of Subclause 10.13, "Construction of an XML element", are applied with **EN** as *QNAME*, **ENSURI** as *NAMESPACE*, **ATTRS** as *ATTRIBUTES*, **CON** as *CONTENTS*, *OPT* as *OPTION*, and *ENC* as *ENCODING*; let result of <XML element> be the *XMLELEM* returned from the application of those General Rules.

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, "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.

9) Without Feature X085, "Predefined namespace prefixes", conforming SQL language shall not contain an <XML element name> E that has an XML QName prefix that is not equivalent to an <XML namespace prefix> contained in one or more <XML namespace declaration>s that are the scope of the <XML element> that contains E.

10) Without Feature X085, “Predefined namespace prefixes”, conforming SQL language shall not contain an explicit or implicit <XML attribute name> *A* that has an XML QName prefix other than **'xml'** that is not equivalent to an <XML namespace prefix> contained in one or more <XML namespace declaration>s that are the scope of the <XML element> that contains *A*.

6.15 <XML forest>

Function

Generate an XML value with a sequence 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 data type, or a distinct type.

   c) Case:

      i) If F_i contains a <forest element name> FEN_i, then let $\mathbf{FNC}_i$ 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. The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with FN_i as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let $\mathbf{ANC}_i$ be the *XMLNAME* returned from the application of those General Rules.

   d) $\mathbf{FNC}_i$ shall be an XML 1.1 QName.

e) The Syntax Rules of Subclause 10.12, "Determination of namespace URI", are applied with ***FNC***$_i$ as *QNAME* and the <XML forest> as *BNFTERM*; let ***NSURI***$_i$ be the *NSURI* returned from the application of those Syntax Rules.

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

4) If <XML returning clause> is not specified, then it is implementation-defined whether RETURNING CONTENT or RETURNING SEQUENCE is implicit.

5) If <XML namespace declaration> is specified, then let *XND* be "<XML namespace declaration> <comma>"; otherwise, let *XND* be a zero-length string.

6) Let *FEL* be the <forest element list>.

7) Let *XCO* be the specified or implied <XML content option>.

8) Case:

a) If RETURNING CONTENT is specified or implied, then <XML forest> is equivalent to

```
XMLDOCUMENT (
  XMLFOREST (
    XND FEL XCO
    RETURNING SEQUENCE )
  RETURNING CONTENT )
```

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

NOTE 34 — If RETURNING CONTENT is specified or implied, then the declared type of <XML forest> is effectively established by the Syntax Rules of Subclause 6.13, "<XML document>".

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 6.5, "<cast specification>", 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) For each *i* between 1 (one) and *n*, the General Rules of Subclause 10.13, "Construction of an XML element", are applied with $\mathbf{FNC}_i$ as *QNAME*, $\mathbf{NSURI}_i$ as *NAMESPACE*, the empty XQuery sequence as *ATTRIBUTES*, $\mathbf{CON}_i$ as *CONTENTS*, the explicit or implicit <XML content option> as *OPTION*, and *ENC* as *ENCODING*; let XV_i be the *XMLELEM* returned from the application of those General Rules.

6) Let R_1 be XV_1. For all *i*, $2 \leq i \leq n$, the General Rules of Subclause 10.14, "Concatenation of two XML values", are applied with R_{i-1} as *FIRSTVAL* and XV_i as *SECONDVAL*; let R_i be the *CONCAT* returned from the application of those General Rules.

7) 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, "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) Without Feature X085, "Predefined namespace prefixes", conforming SQL language shall not contain an explicit or implicit <forest element name> *F* that has an XML QName prefix that is not equivalent to an <XML namespace prefix> contained in one or more <XML namespace declaration>s that are the scope of the <XML forest> that contains *F*.

6.16 <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) Case:

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

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

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 let *PARSE* be the null value.

   b) Otherwise, the General Rules of Subclause 10.16, "Parsing a string as an XML value", are applied with *DC* as *SYNTAX*, *V* as *TEXT*, and *WO* as *OPTION*; let *PARSE* be the *XMLPARSE* returned from the application of those General Rules.

5) The result of <XML parse> is *PARSE*.

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

4) Without Feature X066, “XMLParse: binary 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 binary string type and <XML parse> shall not immediately contain a <document or content> that is DOCUMENT.

6.17 <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> <character 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 NCName.

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 it is implementation-defined whether RETURNING CONTENT or RETURNING SEQUENCE is implicit.

5) Let *XPT* be the <XML PI target>.

6) If <character value expression> is specified, then let *CSVE* be "<comma> <character value expression>"; otherwise, let *CSVE* be the zero-length string.

7) Case:

   a) If RETURNING CONTENT is specified or implied, then <XML PI> is equivalent to:

```
XMLDOCUMENT (
  XMLPI (
    NAME XPT CSVE
    RETURNING SEQUENCE )
  RETURNING CONTENT )
```

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

NOTE 35 — If RETURNING CONTENT is specified or implied, then the declared type of <XML PI> is effectively established by the Syntax Rules of Subclause 6.13, "<XML document>".

Access Rules

None.

General Rules

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

2) If <character 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>.

3) The General Rules of Subclause 9.8, "Mapping values of SQL data types to values of XML Schema data types", are applied with *SVE* as *SQLVALUE*, an indication that binary strings are to be encoded in hex as *ENCODING*, "absent" as *NULLS*, and *False* as *CHARMAPPING*; let *USV* be the *XMLVALUE* returned from the application of those General Rules. *USV* is a character string of Unicode characters.

4) If the value of

   `'<?' I || ' ' || USV || '?>'`

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

   NOTE 36 — 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].

5) The Syntax Rules of Subclause 10.21, "Determination of an XQuery formal type notation", are applied with *USV* as *SOURCE* and *False* as *CONTEXT*; let ***XVFT*** be the *FORMALTYPE* returned from the application of those Syntax Rules.

6) The Syntax Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied with the null value as *NONTERMINAL*; let ***XSC*** be the *STATICCONTEXT* returned from the application of those Syntax Rules.

   NOTE 37 — ***XSC*** is an XQuery static context.

7) The General Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied; let ***XDC*** be the *DYNAMICCONTEXT* returned from the application of those General Rules.

   NOTE 38 — ***XDC*** is an XQuery dynamic context.

8) Let *XSC* and *XDC* be augmented with an XQuery variable **$EXPR**, whose XQuery formal type notation is ***XVFT***, and whose value is ***USV***.

9) Case:

   a) If Feature X211, "XML 1.1 support" is supported, then let ***PN*** 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

      ```
      processing-instruction I { $EXPR }
      ```

   b) Otherwise, let ***PN*** be the result of an XQuery evaluation with XML 1.0 lexical rules, using ***XSC*** and ***XDC*** as the XQuery expression context of the XQuery expression

      ```
      processing-instruction I { $EXPR }
      ```

10) The result of the <XML PI> is ***PN***.

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, an <identifier> contained in an <XML PI target>, when mapped to Unicode, shall be an XML 1.0 NCName.

   NOTE 39 — The set of XML 1.0 NCNames is a proper subset of the set of XML 1.1 NCNames. That is, all XML 1.0 NCNames are also XML 1.1 NCNames, but not all XML 1.1 NCNames are also XML 1.0 NCNames.

3) Without Feature X241, “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.

6.18 <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> ] ]
       <XML query empty handling option>
       <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>

<XML query empty handling option> ::=
    NULL ON EMPTY
  | EMPTY ON EMPTY
```


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) If <XML returning clause> is not specified, then it is implementation-defined whether RETURNING CONTENT or RETURNING SEQUENCE is implicit.

4) Case:

   a) If the implicit or explicit <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*.

5) The Syntax Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied with the <XML query> as *NONTERMINAL*; let ***XSC*** be the *STATICCONTEXT* returned from the application of those Syntax Rules, modifying the **in-scope variables** component of ***XSC*** as follows:

   NOTE 40 — ***XSC*** is an XQuery static context.

   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 6.6, "<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) If *XQV* contains an <XML passing mechanism>, then let *VPM* be that <XML passing mechanism>; otherwise, let *VPM* be *DPM*.

      vii) The Syntax Rules of Subclause 10.21, "Determination of an XQuery formal type notation", are applied with *VVE* as *SOURCE* and *False* as *CONTEXT*; let ***VFT*** be the *FORMALTYPE* returned from the application of those Syntax Rules.

      viii) The pair (*I*, ***VFT***) is placed in the **in-scope variables** component 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 6.6, "<XML cast specification>".

      iv) If the 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) The Syntax Rules of Subclause 10.21, "Determination of an XQuery formal type notation", are applied with *CVE* as *SOURCE* and *True* as *CONTEXT*; let ***CFT*** be the *FORMALTYPE* returned from the application of those Syntax Rules.

vii) Let **$*fs*:dot** be the XQuery variable in the fictitious namespace associated with the prefix ***fs*** as posited in [XQueryFS], 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 41 — Initialization of the XQuery static context ***XSC*** can in many cases be overridden by the prolog of the <XQuery expression>.

6) <XQuery expression> shall conform to an implementation-defined subset of the normative rules of XQuery expressions that are found in [XQuery], [XQueryFO], [XQueryFS], and [XQuery Update], and that define the analysis phase, using the XQuery static context *XSC* and XML 1.1 lexical rules, without raising an XQuery static error, type error, or statically-detected dynamic error.

NOTE 42 — The subset of mandatory and optional features of XQuery supported by an implementation is implementation-defined. This includes:

— What subset of the BNF in [XQuery] and [XQuery Update] is supported.

— For the supported BNF, which of the normative rules in [XQuery] and [XQueryFS] are supported.

— Which of the functions in [XQueryFO] are supported.

— Which of the features described in the [XQuery] section titled "Optional Features" are supported, and to what extent.

NOTE 43 — XQuery permits an implementation to postpone some or all type checking to the evaluation phase (*i.e.*, until the General Rules of this Subclause are performed).

7) <XQuery expression> shall not contain an updating expression, except as part of a transform expression, as defined by [XQuery Update].

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

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

10) Let *XQEHO* be the <XML query empty handling option>.

11) Case:

a) If RETURNING CONTENT is specified or implied, then <XML query> is equivalent to

```
XMLDOCUMENT (
  XMLQUERY (
    XQE XQAL
    RETURNING SEQUENCE XQRM XQEHO )
  RETURNING CONTENT )
```

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

NOTE 44 — If RETURNING CONTENT is specified or implied, then the declared type of <XML query> is effectively established by the Syntax Rules of Subclause 6.13, "<XML document>".

Access Rules

None.

General Rules

1) The General Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied; let ***XDC*** be the *DYNAMICCONTEXT* returned from the application of those General Rules.

NOTE 45 — ***XDC*** is an XQuery dynamic context.

a) Case:

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

ii) Otherwise,

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

Case:

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

2) Otherwise, the General Rules of Subclause 10.18, "Constructing a copy of an XML value", are applied with ***CVE*** as *VALUE*; let ***CI*** be the *COPY* returned from the application of those General Rules.

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

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

B) Case:

I) If *CI* is the empty XQuery sequence, then there is no context item in ***XDC***.

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*.

b) 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

Case:

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

B) Otherwise, the General Rules of Subclause 10.18, "Constructing a copy of an XML value", are applied with ***V*** as *VALUE*; let ***VV*** be the *COPY* returned from the application of those General Rules.

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 *X1* is an empty sequence and the <XML query empty handling option> is NULL ON EMPTY, then let *X2* be the null value.

b) Otherwise,

Case:

i) If *XQRM* is BY REF, then let *X2* be *X1*.

ii) Otherwise, the General Rules of Subclause 10.18, "Constructing a copy of an XML value", are applied with ***X1*** as *VALUE*; let ***X2*** be the *COPY* returned from the application of those General Rules.

5) *X2* is the result of the <XML query>.

Conformance Rules

1) Without Feature X200, "XMLQuery", conforming SQL language shall not contain an <XML query>.

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

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

4) Without Feature X203, "XMLQuery: passing a context item", in conforming SQL language, an <XML query> shall not contain an <XML query context item>.

5) Without Feature X204, “XMLQuery: initializing an XQuery variable”, in conforming SQL language, an <XML query> shall not contain an <XML query variable>.

6) Without Feature X205, “XMLQuery: EMPTY ON EMPTY option”, conforming SQL language shall not contain an <XML query> that contains an <XML query empty handling option> that specifies EMPTY ON EMPTY.

7) Without Feature X206, “XMLQuery: NULL ON EMPTY option”, conforming SQL language shall not contain an <XML query> that contains an <XML query empty handling option> that specifies NULL ON EMPTY.

8) 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.

9) 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.

6.19 <XML text>

Function

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

Format

```
<XML text> ::=
  XMLTEXT <left paren> <character value expression>
      [ <XML returning clause> ] <right paren>
```

Syntax Rules

1) If <XML returning clause> is not specified, then it is implementation-defined whether RETURNING CONTENT or RETURNING SEQUENCE is implicit.

2) Let *SVE* be the <character value expression>.

3) Case:

   a) If RETURNING CONTENT is specified or implied, then <XML text> is equivalent to

```
XMLDOCUMENT (
  XMLTEXT ( SVE RETURNING SEQUENCE )
  RETURNING CONTENT )
```

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

NOTE 46 — If RETURNING CONTENT is specified or implied, then the declared type of <XML text> is effectively established by the Syntax Rules of Subclause 6.13, "<XML document>".

Access Rules

None.

General Rules

1) Let *SV* be the value of *SVE*.

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

3) Case:

   a) If <XML text> 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 text>.

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

4) Case:

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

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

5) The General Rules of Subclause 9.8, "Mapping values of SQL data types to values of XML Schema data types", are applied with *SV* as *SQLVALUE*, *ENC* as *ENCODING*, "absent" as *NULLS*, and *False* as *CHARMAPPING*; let ***USV*** be the *XMLVALUE* returned from the application of those General Rules. ***USV*** is a character string of Unicode characters.

6) Let ***XTN*** be an XQuery text node with the following properties:

a) The value of the **content** property is the value of ***USV***.

b) The value of the **parent** property is set to **empty**.

7) The result of <XML text> is the XQuery sequence consisting of one node, ***XTN***.

Conformance Rules

1) Without Feature X038, "XMLText", conforming SQL language shall not contain an <XML text>.

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

6.20 <XML validate>

Function

Validate an XML value.

Format

```
<XML validate> ::=
  XMLVALIDATE <left paren>
      <document or content or sequence>
      <XML value expression>
      [ <XML valid according to clause> ]
      <right paren>

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

Syntax Rules

1) Let *XVE* be the <XML value expression>. Let *DCS* be the <document or content or sequence>.

2) If <XML valid according to clause> is specified, then let *RXS* be the indicated registered XML Schema, let *ENSURI* be the indicated XML namespace, if any, and let *EN* be the XML NCName of the indicated global element declaration schema component, if any.

   NOTE 47 — Indicated registered XML Schema, indicated XML namespace, and indicated global element declaration schema component are defined in Subclause 11.6, "<XML valid according to clause>".

3) Let *XVT* be the declared type of *XVE*. The declared type of <XML validate> is

   Case:

   a) If *XVT* is XML(DOCUMENT(ANY)), XML(DOCUMENT(UNTYPED)), or XML(DOCUMENT(XMLSCHEMA)), then

      Case:

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

         Case:

         1) If <XML valid element name specification> is specified, then XML(DOCUMENT(XMLSCHEMA)) whose XML type descriptor includes the registered XML Schema descriptor of *RXS*, the XML namespace *ENSURI*, and XML NCName *EN*.

         2) If <XML valid element namespace specification> is specified, then XML(DOCUMENT(XMLSCHEMA)) whose XML type descriptor includes the registered XML Schema descriptor of *RXS* and the XML namespace *ENSURI*.

         3) Otherwise, XML(DOCUMENT(XMLSCHEMA)) whose XML type descriptor includes the registered XML Schema descriptor of *RXS*.

      ii) Otherwise, XML(DOCUMENT(ANY)).

b) If *XVT* is XML(CONTENT(ANY)), XML(CONTENT(UNTYPED)), or XML(CONTENT(XMLSCHEMA)), then

Case:

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

Case:

1) If <XML valid element name specification> is specified, then XML(CONTENT(XMLSCHEMA)) whose XML type descriptor includes the registered XML Schema descriptor of *RXS*, the XML namespace *ENSURI*, and XML NCName *EN*.

2) If <XML valid element namespace specification> is specified, then XML(CONTENT(XMLSCHEMA)) whose XML type descriptor includes the registered XML Schema descriptor of *RXS* and the XML namespace *ENSURI*.

3) Otherwise, XML(CONTENT(XMLSCHEMA)) whose XML type descriptor includes the registered XML Schema descriptor of *RXS*.

ii) Otherwise, XML(CONTENT(ANY)).

c) If *XVT* is XML(SEQUENCE), then XML(SEQUENCE).

Access Rules

None.

General Rules

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

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

3) Case:

a) If *DCS* is DOCUMENT and *V* is not an XQuery sequence of length 1 (one) whose sole XQuery item is an XQuery document node whose **children** property contains exactly one XQuery element node, zero or more XQuery comment nodes, and zero or more XQuery processing instruction nodes, then an exception condition is raised: *data exception — invalid XML document*.

b) If *DCS* is CONTENT and *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 — invalid XML content*.

c) Otherwise, if any XQuery item contained in *V* is an XQuery atomic value, XQuery attribute node, XQuery namespace node, or an XQuery text node, then an exception condition is raised: *data exception — XQuery sequence cannot be validated*.

4) If *V* is an empty XQuery sequence, then the result of <XML validate> is an empty XQuery sequence and no further General Rules of this Subclause are applied.

5) Let *N* be the number of XQuery items in *V*. Let $\boldsymbol{I}_j$, 1 (one) ≤ *j* ≤ *N*, be an enumeration in order of the XQuery items in *V*.

6) For each *j*, 1 (one) ≤ *j* ≤ *N*,

Case:

a) If $\boldsymbol{I}_j$ is an XQuery document node ***D***, then:

i) Let *M* be the number of XQuery nodes in the **children** property of ***D***.

ii) Let $\boldsymbol{C}_k$, 1 (one) ≤ *k* ≤ *M*, be an enumeration in order of the XQuery nodes in the **children** property of ***D***.

iii) For each *k*, 1 (one) ≤ *k* ≤ *M*,

Case:

1) If $\boldsymbol{C}_k$ is not an XQuery element node, an XQuery comment node, or an XQuery processing instruction node, then an exception condition is raised: *data exception — XQuery document node cannot be validated*.

2) If $\boldsymbol{C}_k$ is an XQuery element node, then:

A) Case:

I) If <XML valid according to clause> is specified, then let XS_k be *RXS*.

II) Otherwise:

1) Let ***NS*** be the XML namespace of $\boldsymbol{C}_k$.

2) Let XS_k be a registered XML Schema for which the current user has USAGE privilege and whose target namespace is identical to ***NS***, 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 no XS_k is found, then an exception condition is raised: *data exception — no XML schema found*.

B) If <XML valid element clause> is specified, then let ***ENAME*** be the **node-name** property of $\boldsymbol{C}_k$.

Case:

I) If <XML valid element name specification> is specified, and either the XML NCName of ***ENAME*** is not equivalent to *EN*, or the XML namespace of ***ENAME*** is not identical to *ENSURI*, as defined by [Namespaces], then an exception condition is raised: *data exception — no XML element with the specified QName*.

II) Otherwise, if <XML valid element namespace specification> is specified, and the XML namespace of ***ENAME*** is not identical to *ENSURI*, as defined by [Namespaces], then an exception condition is raised: *data exception — no XML element with the specified namespace*.

C) Case:

I) If *DCS* is DOCUMENT, then the General Rules of Subclause 10.22, "Validating an XQuery document or element node", are applied with $\boldsymbol{I}_j$ as *ITEM* and XS_k as *SCHEMA*. Let *STAT* be the *STATUS* and ***RES*** be the *RESULT*, if any, returned by the General Rules of Subclause 10.22, "Validating an XQuery document or element node".

II) Otherwise, the General Rules of Subclause 10.22, "Validating an XQuery document or element node", are applied with $\boldsymbol{C}_k$ as *ITEM* and XS_k as *SCHEMA*. Let *STAT* be the *STATUS* and ***RES*** be the *RESULT*, if any, returned by the General Rules of Subclause 10.22, "Validating an XQuery document or element node".

D) If *STAT* is FAILURE, then an exception condition is raised: *data exception — validation failure*.

E) Case:

I) If *DCS* is DOCUMENT, then let $\boldsymbol{O}_k$ be the XQuery element node that is the **child** of ***RES***.

II) Otherwise, let $\boldsymbol{O}_k$ be ***RES***.

3) Otherwise, the General Rules of Subclause 10.18, "Constructing a copy of an XML value", are applied with $\boldsymbol{C}_k$ as *VALUE*; let $\boldsymbol{O}_k$ be the *COPY* returned from the application of those General Rules.

iv) Let $\boldsymbol{R}_j$ be an XQuery document node whose **children** property is replaced by an enumeration of $\boldsymbol{O}_k$, 1 (one) ≤ k ≤ M.

b) If $\boldsymbol{I}_j$ is an XQuery element node ***E***, then

i) Case:

1) If <XML valid according to clause> is specified, then let *XS* be *RXS*.

2) Otherwise, let ***NS*** be the XML namespace of ***E***. Let *XS* be a registered XML Schema for which the current user has USAGE privilege and whose target namespace is identical to ***NS***, 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 no *XS* is found, then an exception condition is raised: *data exception — no XML schema found*.

ii) If <XML valid element clause> is specified, then let ***ENAME*** be the **node-name** property of ***E***.

Case:

1) If <XML valid element name specification> is specified, and either the XML NCName of ***ENAME*** is not equivalent to *EN* or the XML namespace of ***ENAME*** is not identical to *ENSURI*, as defined by [Namespaces], then an exception condition is raised: *data exception — no XML element with the specified QName*.

2) Otherwise, if <XML valid element namespace specification> is specified and the XML namespace of ***ENAME*** is not identical to *ENSURI*, as defined by [Namespaces], then an

exception condition is raised: *data exception — no XML element with the specified namespace*.

iii) The General Rules of Subclause 10.22, "Validating an XQuery document or element node", are applied with $\boldsymbol{I}_j$ as *ITEM* and *XS* as *SCHEMA*. Let *STAT* be the *STATUS* and ***RES*** be the *RESULT*, if any, returned by the General Rules of Subclause 10.22, "Validating an XQuery document or element node".

iv) If *STAT* is FAILURE, then an exception condition is raised: *data exception — validation failure*.

v) Let $\boldsymbol{R}_j$ be ***RES***.

c) Otherwise, $\boldsymbol{I}_j$ is an XQuery comment node or an XQuery processing instruction node. The General Rules of Subclause 10.18, "Constructing a copy of an XML value", are applied with $\boldsymbol{I}_j$ as *VALUE*; let $\boldsymbol{R}_j$ be the *COPY* returned from the application of those General Rules.

7) Let ***R*** be an XQuery sequence enumerated by $\boldsymbol{R}_j$, 1 (one) ≤ j ≤ N.

8) The result of <XML validate> is ***R***.

Conformance Rules

1) Without Feature X271, "XMLValidate: data-driven case", conforming SQL language shall not contain an <XML validate> that does not contain <XML valid according to clause>.

2) Without Feature X272, "XMLValidate: ACCORDING TO clause", conforming SQL language shall not contain an <XML validate> that contains <XML valid according to clause>.

3) Without Feature X273, "XMLValidate: ELEMENT clause", conforming SQL language shall not contain an <XML validate> that contains <XML valid element clause>.

4) Without Feature X284, "XMLValidate: NAMESPACE without ELEMENT clause", conforming SQL language shall not contain an <XML validate> that contains an <XML valid element clause> that does not contain an <XML valid element name specification>.

5) Without Feature X286, "XMLValidate: NO NAMESPACE with ELEMENT clause", conforming SQL language shall not contain an <XML validate> that contains an <XML valid element namespace specification> that contains NO NAMESPACE.

6) Without Feature X274, "XMLValidate: schema location", conforming SQL language shall not contain an <XML validate> that contains <XML valid schema location>.

7) Without Feature X281, "XMLValidate with DOCUMENT option", conforming SQL language shall not contain an <XML validate> that immediately contains a <document or content or sequence> that is DOCUMENT.

8) Without Feature X282, "XMLValidate with CONTENT option", conforming SQL language shall not contain an <XML validate> that immediately contains a <document or content or sequence> that is CONTENT.

9) Without Feature X283, "XMLValidate with SEQUENCE option", conforming SQL language shall not contain an <XML validate> that immediately contains a <document or content or sequence> that is SEQUENCE.

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 argument list> ]
      COLUMNS <XML table column definitions> <right paren>

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

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

<XML table argument passing mechanism> ::=
  <XML passing mechanism>

<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) Insert after SR 20)a) If <table primary> specifies <XML table> or <XML iterate>, then the <table primary> is not generally updatable, not simply updatable, not effectively updatable, and not insertable-into.

2) Insert after SR 27) An <XML table> is *possibly non-deterministic* if it does not conform to implementation-defined rules enabling the SQL-implementation to deduce that the result of the <XML table> is deterministic.

3) Insert after SR 27) A <table primary> is also *possibly non-deterministic* if it is an <XML table> that is possibly non-deterministic.

4) 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).

5) 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 argument list> is specified, then let *XTTPM* be the <XML table argument passing mechanism>, 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, and let *XQAL* be

         `PASSING` *XTTPM* XQA_1`, ...,` XQA_{NA}

      ii) Otherwise, let *XTTPM* be BY REF, and 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
```

ii) Otherwise:

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

2) Case:

A) If DT_j is XML(SEQUENCE) then:

I) If <XML table argument list> is not specified, then $XTCD_j$ shall contain <XML passing mechanism>.

II) If $XTCD_j$ contains <XML passing mechanism>, then let XPM_j be this <XML passing mechanism>; otherwise, let XPM_j be *XTTPM*.

III) 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 a <character string literal> whose value is equivalent to the column name of the column identified by CN_j.

5) Let XQC_j be

```
XMLQUERY ( PATH_j PASSING BY REF I.V XRM_j EMPTY ON EMPTY )
```

6) Let XE_j be

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

7) Case:

A) If DT_j is XML(SEQUENCE), then let CPM_j be XPM_j.

B) If DT_j is an XML type, then let CPM_j be BY VALUE.

C) Otherwise, let CPM_j be a zero-length string.

8) Let SLI_j be

```
CASE WHEN XE_j
     THEN XMLCAST( XQC_j AS DT_j CPM_j )
     ELSE DEF_j
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 *DCLP* be the zero-length string.

h) The <XML table> is equivalent to

```
LATERAL
  ( XNDC
    SELECT SLI_1 AS CN_1, SLI_2 AS CN_2, ..., SLI_NC AS CN_NC
    FROM XMLITERATE ( XMLQUERY ( XTRP XQAL
                      RETURNING SEQUENCE BY REF EMPTY ON EMPTY ) )
         AS I ( V, N )
  ) AS CORR DCLP
```

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 3) 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>.

NOTE 48 — This Conformance Rule ensures that <XML iterate> exists purely as a specification device within this edition of this part of this International Standard and cannot be used by conforming SQL language. Conforming SQL language may achieve the same effect by use of <XML table>.

2) Insert this CR Without Feature X300, “XMLTable”, conforming SQL language shall not contain <XML table>.

3) Insert this CR Without Feature X301, “XMLTable: derived column list option”, in conforming SQL language, a <table primary> that is an <XML table> shall not contain a <derived column list>.

4) Insert this CR Without Feature X302, “XMLTable: ordinality column option”, in conforming SQL language, an <XML table> shall not contain an <XML table ordinality column definition>.

5) Insert this CR Without Feature X303, “XMLTable: column default option”, in conforming SQL language, an <XML table regular column definition> shall not contain a <default clause>.

6) Insert this CR Without Feature X304, “XMLTable: passing a context item”, in conforming SQL language, an <XML table argument list> shall not contain an <XML query context item>.

7) Insert this CR Without Feature X305, “XMLTable: initializing an XQuery variable”, in conforming SQL language, an <XML table argument list> shall not contain an <XML query variable>.

8) Insert this CR Without Feature X086, “XML namespace declarations in XMLTable”, in conforming SQL language, an <XML table> shall not contain an <XML namespace declaration>.

7.2 <query expression>

This Subclause modifies Subclause 7.17, "<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

1) Replace GR 3)b)ii)1) If the declared type of the *i*-th column of *T* is not an XML type, then let DTC_i be that declared type; otherwise, let $DDTC_i$ be the declared type of the *i*-th column of *T* and let DTC_i be

   $DDTC_i$ BY REF

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 <query expression> that is contained in another <query expression> 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 exists predicate>
  | <XML valid 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>, <regex like predicate>, <null predicate>, <quantified comparison predicate>, <exists predicate>, <unique predicate>, <normalized predicate>, <match predicate>, <overlaps predicate>, <distinct predicate>, <member predicate>, <submultiset predicate>, <set predicate>, <type predicate>, <period predicate>, <JSON predicate>, <JSON exists predicate>, <XML content predicate>, <XML document predicate>, <XML exists predicate>, or <XML valid predicate>.

Conformance Rules

No additional Conformance Rules.

8.2 <XML content predicate>

Function

Determine whether an XML value is an XQuery document node.

Format

```
<XML content predicate> ::=
  <row value predicand> <XML content predicate part 2>

<XML content predicate part 2> ::=
  IS [ NOT ] CONTENT
```

Syntax Rules

1) The <row value predicand> shall be a <row value constructor predicand> that is a <common value expression> that is an <XML value expression> *XVE*.

2) The expression

   XVE IS NOT CONTENT

   is equivalent to

   NOT (*XVE* IS CONTENT)

Access Rules

None.

General Rules

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

2) The result of

   XVE IS CONTENT

   is determined as follows.

   Case:

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

   b) If *V* is an XQuery sequence of length 1 (one) whose sole XQuery item is an XQuery document node, 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>.

8.3 <XML document predicate>

Function

Determine whether an XML value is an XQuery document node whose **children** property contains exactly one XQuery element node, zero or more XQuery comment nodes, and zero or more XQuery processing instruction nodes.

Format

```
<XML document predicate> ::=
  <row value predicand> <XML document predicate part 2>

<XML document predicate part 2> ::=
  IS [ NOT ] DOCUMENT
```

Syntax Rules

1) The <row value predicand> shall be a <row value constructor predicand> that is a <common value expression> that is an <XML value expression> *XVE*.

2) The expression

   XVE IS NOT DOCUMENT

   is equivalent to

   NOT (*XVE* IS DOCUMENT)

Access Rules

None.

General Rules

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

2) The result of

   XVE IS DOCUMENT

   is determined as follows.

   Case:

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

   b) If *V* is an XQuery sequence of length 1 (one) whose sole XQuery item is an XQuery document node whose **children** property contains exactly one XQuery element node, zero or more XQuery comment nodes, and zero or more XQuery processing instruction nodes, 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>.

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 6.18, "<XML query>", are applied to

   ```
   XMLQUERY ( XQE XQP RETURNING SEQUENCE EMPTY ON EMPTY )
   ```

Access Rules

1) The Access Rules of Subclause 6.18, "<XML query>", are applied to

   ```
   XMLQUERY ( XQE XQP RETURNING SEQUENCE EMPTY ON EMPTY )
   ```

General Rules

1) Let *V* be the value of

   ```
   XMLQUERY ( XQE XQP RETURNING SEQUENCE EMPTY ON EMPTY )
   ```

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.

Format

```
<XML valid predicate> ::=
  <row value predicand> <XML valid predicate part 2>

<XML valid predicate part 2> ::=
  IS [ NOT ] VALID
      <document or content or sequence>
      [ <XML valid according to clause> ]
```

Syntax Rules

1) The <row value predicand> shall be a <row value constructor predicand> that is a <common value expression> that is an <XML value expression> *XVE*.

2) Let *DCS* be the <document or content or sequence>. Let *XVACC* be the <XML valid according to clause>, if any; otherwise, let *XVACC* be the zero-length string.

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

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

4) If <XML valid according to clause> is specified, then let *RXS* be the indicated registered XML Schema, let *ENSURI* be the indicated XML namespace, if any, and let *EN* be the XML NCName of the indicated global element declaration schema component, if any.

   NOTE 49 — Indicated registered XML Schema, indicated XML namespace, and indicated global element declaration schema component are defined in Subclause 11.6, "<XML valid according to clause>".

Access Rules

None.

General Rules

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

2) If *V* is the null value, then the result of the <XML valid predicate> is *Unknown* and no further General Rules of this Subclause are applied.

3) If *DCS* is DOCUMENT and *V* is not an XQuery sequence of length 1 (one) whose sole XQuery item is an XQuery document node whose **children** property contains exactly one XQuery element node, zero or more XQuery comment nodes, and zero or more XQuery processing instruction nodes, then the result of the <XML valid predicate> is *False* and no further General Rules of this Subclause are applied.

4) If *DCS* is CONTENT and *V* is not an XQuery sequence of length 1 (one) whose sole XQuery item is an XQuery document node, then the result of the <XML valid predicate> is *False* and no further General Rules of this Subclause are applied.

5) If *DCS* is SEQUENCE, then

Case:

a) If *V* is an empty XQuery sequence, then the result of the <XML valid predicate> is *True* and no further General Rules of this Subclause are applied.

b) Otherwise, if any XQuery item contained in *V* is an XQuery atomic value, XQuery attribute node, XQuery namespace node, or an XQuery text node, then the result of the <XML valid predicate> is *False* and no further General Rules of this Subclause are applied.

6) Let *N* be the number of XQuery items in *V*. Let $\boldsymbol{I}_j$, 1 (one) ≤ *j* ≤ *N*, be an enumeration in order of the XQuery items in *V*. Let $TV_{0\ (zero)}$ be *True*.

7) For each *j*, 1 (one) ≤ *j* ≤ *N*,

Case:

a) If $\boldsymbol{I}_j$ is an XQuery document node ***D***, then:

i) Let *M* be the number of XQuery nodes in the **children** property of ***D***.

ii) Let $\boldsymbol{C}_k$, 1 (one) ≤ *k* ≤ *M*, be an enumeration in order of the XQuery nodes in the **children** property of ***D***. Let $ITV_{0\ (zero)}$ be *True*.

iii) For each *k*, 1 (one) ≤ *k* ≤ *M*,

Case:

1) If $\boldsymbol{C}_k$ is not an XQuery element node, XQuery comment node, or an XQuery processing instruction node, then the result of the <XML valid predicate> is *False* and no further General Rules of this Subclause are applied.

2) If $\boldsymbol{C}_k$ is an XQuery comment node or an XQuery processing instruction node, then let ITV_k be ITV_{k-1}.

3) Otherwise, $\boldsymbol{C}_k$ is an XQuery element node.

A) Case:

I) If <XML valid according to clause> is specified, then let XS_k be *RXS* and let *FOUND* be *True*.

II) Otherwise:

1) Let ***NS*** be the XML namespace of $\boldsymbol{C}_k$.

2) Let XS_k be a registered XML Schema for which the current user has USAGE privilege and whose target namespace is identical to ***NS***, 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 there is no XS_k, then let *FOUND* be *False*; otherwise, let *FOUND* be *True*.

B) Case:

I) If *FOUND* is *False*, then let ITV_k be the result of

ITV_{k-1} AND *Unknown*

II) Otherwise:

1) If <XML valid element clause> is specified, then let ***ENAME*** be the **node-name** property of C_k.

Case:

a) If <XML valid element name specification> is specified, and if either the XML NCName of ***ENAME*** is not equivalent to *EN* or the XML namespace of ***ENAME*** is not identical to *ENSURI*, as defined by [Namespaces], then the result of the <XML valid predicate> is *False* and no further General Rules of this Subclause are applied.

b) Otherwise, if <XML valid element namespace specification> is specified, and the XML namespace of ***ENAME*** is not identical to *ENSURI*, as defined by [Namespaces], then the result of the <XML valid predicate> is *False* and no further General Rules of this Subclause are applied.

2) Case:

a) If *DCS* is DOCUMENT, then the General Rules of Subclause 10.22, "Validating an XQuery document or element node", are applied with $\boldsymbol{I}_j$ as *ITEM* and XS_k as *SCHEMA*. Let *STAT* be the *STATUS* returned by the General Rules of Subclause 10.22, "Validating an XQuery document or element node".

b) Otherwise, the General Rules of Subclause 10.22, "Validating an XQuery document or element node", are applied with $\boldsymbol{C}_k$ as *ITEM* and XS_k as *SCHEMA*. Let *STAT* be the *STATUS* returned by the General Rules of Subclause 10.22, "Validating an XQuery document or element node".

3) Case:

a) If *STAT* is FAILURE, then the result of the <XML valid predicate> is *False* and no further General Rules of this Subclause are applied.

b) Otherwise, let ITV_k be ITV_{k-1}.

iv) Let TV_j be the result of

TV_{j-1} AND ITV_M

b) If ***I***$_j$ is an XQuery element node ***E***, then:

i) Case:

1) If <XML valid according to clause> is specified, then let *XS* be *RXS*, and let *FOUND* be *True*.

2) Otherwise, let ***NS*** be the XML namespace of ***E***. Let *XS* be a registered XML Schema for which the current user has USAGE privilege and whose target namespace is identical to ***NS***, 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 there is no *XS*, then let *FOUND* be *False*; otherwise, let *FOUND* be *True*.

ii) Case:

1) If *FOUND* is *False*, then let TV_j be the result of

```
TV(j-1) AND Unknown
```

2) Otherwise:

A) If <XML valid element clause> is specified, then let ***ENAME*** be the **node-name** property of ***E***.

Case:

I) If <XML valid element name specification> is specified, and if either the XML NCName of ***ENAME*** is not equivalent to *EN*, or the XML namespace of ***ENAME*** is not identical to *ENSURI*, as defined by [Namespaces], then the result of the <XML valid predicate> is *False* and no further General Rules of this Subclause are applied.

II) Otherwise, if <XML valid element namespace specification> is specified, and the XML namespace of ***ENAME*** is not identical to *ENSURI*, as defined by [Namespaces], then the result of the <XML valid predicate> is *False* and no further General Rules of this Subclause are applied.

B) The General Rules of Subclause 10.22, "Validating an XQuery document or element node", are applied with ***I***$_j$ as *ITEM* and *XS* as *SCHEMA*. Let *STAT* be the *STATUS* returned by the General Rules of Subclause 10.22, "Validating an XQuery document or element node".

C) Case:

I) If *STAT* is FAILURE, then the result of the <XML valid predicate> is *False* and no further General Rules of this Subclause are applied.

II) Otherwise, let TV_j be TV_{j-1}.

c) Otherwise, ***I***$_j$ is an XQuery comment node or an XQuery processing instruction node. Let TV_j be TV_{j-1}.

8) The result of <XML valid predicate> is TV_N.

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 X155, “IS VALID predicate: NAMESPACE without ELEMENT clause”, conforming SQL language shall not contain an <XML valid predicate> that contains an <XML valid element clause> that does not contain an <XML valid element name specification>.

3) Without Feature X157, “IS VALID predicate: NO NAMESPACE with ELEMENT clause”, conforming SQL language shall not contain an <XML valid predicate> that contains an <XML valid element namespace specification> that contains NO NAMESPACE.

4) 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>.

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

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

7) 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>.

8) Without Feature X151, “IS VALID predicate: with DOCUMENT option”, conforming SQL language shall not contain an <XML valid predicate> that immediately contains a <document or content or sequence> that is DOCUMENT.

9) Without Feature X152, “IS VALID predicate: with CONTENT option”, conforming SQL language shall not contain an <XML valid predicate> that immediately contains a <document or content or sequence> that is CONTENT.

10) Without Feature X153, “IS VALID predicate: with SEQUENCE option”, conforming SQL language shall not contain an <XML valid predicate> that immediately contains a <document or content or sequence> that is SEQUENCE.

9 Mappings

9.1 Mapping SQL <identifier>s to XML Names

Subclause Signature

```
"Mapping SQL <identifier>s to XML Names" [General Rules] (
  Parameter: "IDENT",
  Parameter: "ESCAPE VARIANT"
) Returns: "XMLNAME"
```

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 the *IDENT* and let *EV* be the *ESCAPE VARIANT* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XMLN*, which is returned as *XMLNAME*.

NOTE 50 — *EV* is either *partially escaped* or *fully escaped*. *SQLI* is a sequence of characters of SQL_TEXT.

2) 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.

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

NOTE 51 — Unicode scalar values in the ranges U+0000 through U+001F (inclusive), sometimes called the "C0 controls", and U+007F through U+009F (inclusive), sometimes called "delete" (U+007F) and the "C1 controls" (the remainder of that latter range) are not encoding of abstract characters in Unicode. Programs that conform to the Unicode Standard may treat these Unicode scalar values in exactly the same way as they treat the 7- and 8-bit equivalents in other protocols. Such usage

constitutes a higher-level protocol and is beyond the scope of the Unicode standard. These Unicode scalar values do not occur in XML Names, but may appear in other places in XML text.

4) For each *i* between 1 (one) and *N*, let T_i be the mapping of S_i to Unicode using *TM* and let X_i be the Unicode character string defined by the following rules.

Case:

a) If S_i has no mapping to Unicode (*i.e.*, $TM(S_i)$ is undefined), then X_i is implementation-defined.

b) If S_i is <colon>, then

Case:

i) If i = 1 (one), then let X_i be `_x003A_`.

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

iii) Otherwise, let X_i be T_i.

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

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

Case:

i) If S_1 is the lowercase letter "x", then let $\boldsymbol{X}_1$ be `_x0078_`.

ii) If S_1 is the uppercase letter "X", then let $\boldsymbol{X}_1$ be `_x0058_`.

e) If 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 52 — This case implies that T_i has a UCS-2 encoding, which is U+$U_5U_6U_7U_8$.

2) Otherwise, let $\boldsymbol{X_i}$ be _x$U_3U_4U_5U_6U_7U_8$_.

f) Otherwise, let X_i be T_i.

NOTE 53 — 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 XML Name that is the character string concatenation of X_1, X_2, ..., and X_N in order from left to right.

Conformance Rules

None.

9.2 Mapping a multi-part SQL name to an XML Name

Subclause Signature

```
"Mapping a multi-part SQL name to an XML Name" [General Rules] (
  Parameter: "SEQUENCE OF SQL IDENTIFIERS"
) Returns: "XMLNAME"
```

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 *SQLIS* be the *SEQUENCE OF SQL IDENTIFIERS* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XMLR*, which is returned as *XMLNAME*.

2) Let *n* be the number of SQL <identifier>s in *SQLIS*. Let $SQLI_i$, 1 (one) ≤ *i* ≤ *n* be an SQL <identifier> equivalent to the *i*-th element of *SQLIS*.

3) 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*.

4) For each *i* between 1 (one) and *n*, the General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with $SQLI_i$ as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let $\boldsymbol{XMLN}_i$ be the *XMLNAME* returned from the application of those General Rules.

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

   $\texttt{NP(}\boldsymbol{XMLN}_1\texttt{) || '.' || NP(}\boldsymbol{XMLN}_2\texttt{) || '.' || ... || NP(}\boldsymbol{XMLN}_n\texttt{)}$

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

Conformance Rules

None.

9.3 Mapping XML Names to SQL <identifier>s

Subclause Signature

```
"Mapping XML Names to SQL <identifier>s" [General Rules] (
  Parameter: "XMLNAME"
) Returns: "SQLIDENT"
```

Function

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

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let ***XMLN*** be the *XMLNAME* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *SQLID*, which is returned as *SQLIDENT*.

2) ***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.

3) 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 Unicode assigned 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 Unicode assigned 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.

4) Let *U* be the Unicode character string constructed by concatenating every U_i, 1 (one) ≤ i ≤ N, in order by i.

5) 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*.

6) Let the SQL <identifier> *SQLID* that is the result of the mapping of ***XMLN*** be the <delimited identifier> **"***SQLI***"**.

Conformance Rules

1) Without Feature X400, “Name and identifier mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard.

9.4 Mapping an SQL data type to an XML Name

Subclause Signature

```
"Mapping an SQL data type to an XML Name" [General Rules] (
  Parameter: "DATA TYPE"
) Returns: "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 *DATA TYPE* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XMLN*, which is returned as *XML NAME*.

   NOTE 54 — *D* is an SQL data type or the underlying data type of a domain.

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 **xs:integer** denoting *MAXCSL*.

   e) Let ***NLIT*** be the canonical XML Schema literal denoting *N* in the lexical representation of XML Schema type **xs: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 **xs: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 **xs: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 **xs: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 **xs: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 **xs: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 **xs: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 **xs: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 **xs: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 ***XTSP*** be the canonical XML Schema literal denoting *TSP* in the lexical representation of XML Schema type **xs: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 **xs: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. The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "Domain", *C*, *S*, and *N* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLN*** be the *XMLNAME* returned from the application of those General Rules.

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. The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "UDT", *C*, *S*, and *N* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLN*** be the *XMLNAME* returned from the application of those General Rules.

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 Schema literal denoting *M* in the lexical representation of XML Schema type **xs: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**.

Conformance Rules

None.

9.5 Mapping SQL data types to XML Schema data types

Subclause Signature

```
"Mapping SQL data types to XML Schema data types" [General Rules] (
  Parameter: "SQLTYPE",
  Parameter: "NULLS",
  Parameter: "ENCODING"
) Returns: "SCHEMA TYPE"
```

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 *SQLTYPE*, let *NULLS* be the *NULLS*, and let *ENCODING* be the *ENCODING* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XMLT*, which is returned as *SCHEMA TYPE*.

   NOTE 55 — *SQLT* is an SQL data type or a domain. *NULLS* specifies the choice of whether to map null values to absent elements ("absent") or to elements that are marked with **`xsi:nil="true"`** ("nil"). *ENCODING* specifies the choice of whether to encode binary strings in base64 or in hex.

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

4) Let **`xs`** 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".

5) 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".

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

7) Case:

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

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

ii) Let *CSM* be the implementation-defined mapping of strings of *SQLCS* to strings of Unicode. Let *MAXCSL* be the maximum length in characters of *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 **xs: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:

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

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

```
<xs: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:

```
<xs: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:


```
<xs: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:

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

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

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

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

i) Let *N* be the length or maximum length of *SQLT*. Let ***NLIT*** be the canonical XML Schema literal denoting *N* in the lexical representation of the XML Schema type **xs: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) Case:

1) If the type designator of *SQLT* is BINARY, then let ***FACET*** be the XML text:

```
<xs:length value="NLIT">
```

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

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

iv) Case:

1) If the type designator of *SQLT* is BINARY, then it is implementation-defined whether the XML text ***ANNT*** is the zero-length string or given by:

```
name="BINARY"
```

2) If the type designator of *SQLT* is VARBINARY, then it is implementation-defined whether the XML text ***ANNT*** is the zero-length string or given by:

```
name="VARBINARY"
```

3) Otherwise, it is implementation-defined whether the XML text ***ANNT*** is the zero-length string or given by:

```
name="BLOB"
```

v) Case:

1) If the type designator of *XQLT* is BINARY, then it is implementation-dependent whether the XML text ***ANNL*** is the zero-length string or given by:

```
length="NLIT"
```

2) Otherwise, 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:

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

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

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


c) If the type designator of *SQLT* is NUMERIC or DECIMAL, then:

i) Let *P* be the precision of *SQLT*. Let ***PLIT*** be the canonical XML Schema literal denoting *P* in the lexical representation of the XML Schema type **xs:integer**. Let ***FACETP*** be the XML text:

```
<xs:totalDigits value="PLIT"/>
```

ii) Let *S* be the scale of *SQLT*. Let ***SLIT*** be the canonical XML Schema literal denoting *S* in the lexical representation of the XML Schema type **xs:integer**. Let ***FACETS*** be the XML text:

```
<xs: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 the canonical XML Schema literal denoting *UP* in the lexical representation of the XML Schema type **xs:integer**. It is implementation-defined whether the XML text ***ANNP*** is the zero-length string or:

```
userPrecision="UPLIT"
```

NOTE 56 — *UP* may be less than *P*, as specified in SR 27) of Subclause 6.1, "<data type>", in [ISO9075-2].

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:

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

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

```
<xs:simpleType>
  ANN
  <xs:restriction base="xs:decimal">
    FACETP
    FACETS
  </xs:restriction>
</xs: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 the canonical XML Schema literal denoting *MAX* in the lexical representation of the XML Schema type **xs:integer**. Let ***FACETMAX*** be the XML text:

```
<xs:maxInclusive value="MAXLIT"/>
```

ii) Let *MIN* be the minimum value representable by *SQLT*. Let ***MINLIT*** be the canonical XML Schema literal denoting *MIN* in the lexical representation of the XML Schema type **xs:integer**. Let ***FACETMIN*** be the XML text:

```
<xs: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:

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

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

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


iv) It is implementation-defined whether *REST* is

```
<xs:restriction base="xs:integer">
  FACETMAX
  FACETMIN
</xs: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 **xs: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:

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

Table 3 — Constraining facets of XML Schema integer types

Type	minInclusive	maxInclusive
xs:unsignedByte	0	255
xs:byte	-128	127
xs:unsigned-Short	0	2^{16}-1 (65,535)
xs:short	-2^{15} (-32,768)	2^{15}-1 (32,767)

Type	minInclusive	maxInclusive
`xs:unsignedInt`	0	2^{32}-1 (4,294,967,295)
`xs:int`	-2^{31} (-2,147,483,648)	2^{31}-1 (2,147,483,647)
`xs:unsignedLong`	0	2^{64}-1 (18,446,744,073,709,551,615)
`xs: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:

```
<xs:simpleType>
  ANN
  REST
</xs: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 the canonical XML Schema literal denoting *UP* in the lexical representation of the XML Schema type `xs:integer`. It is implementation-defined whether the XML text ***ANNUP*** is the zero-length string or:


```
userPrecision="UPLIT"
```

NOTE 57 — *UP* may be less than *P*, as specified in SR 29) of Subclause 6.1, "<data type>", in [ISO9075-2].

iv) Let ***PLIT*** be the canonical XML Schema literal denoting *P* in the lexical representation of the XML Schema type **xs:integer**. It is implementation-defined whether the XML text ***ANNP*** is the zero-length string or:

```
precision="PLIT"
```

v) Let ***MINLIT*** be the canonical XML Schema literal denoting *MINEXP* in the lexical representation of the XML Schema type **xs:integer**. It is implementation-defined whether the XML text ***ANNMIN*** is the zero-length string or:

```
minExponent="MINLIT"
```

vi) Let ***MAXLIT*** be the canonical XML Schema literal denoting *MAXEXP* in the lexical representation of the XML Schema type **xs: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:

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

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

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

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

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

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

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

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

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

i) Let *S* be the <time fractional seconds precision> of *SQLT*. Let ***SLIT*** be the canonical XML Schema literal denoting *S* in the lexical representation of XML Schema type **xs:integer**.

ii) Case:

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

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

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

```
<xs: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:

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

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

```
<xs:simpleType>
```



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

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

  i) Let *S* be the <time fractional seconds precision> of *SQLT*. Let ***SLIT*** be the canonical XML Schema literal denoting *S* in the lexical representation of XML Schema type **xs: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:

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

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

```
<xs: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:

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

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

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

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

i) Let *S* be the <time fractional seconds precision> of *SQLT*. Let ***SLIT*** be the canonical XML Schema literal denoting *S* in the lexical representation of XML Schema type **xs: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:

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

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

```
<xs: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:

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

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

```
<xs:simpleType>
  ANN
  <xs:restriction base="xs:dateTime">
    FACETP
  </xs:restriction>
</xs:simpleType>
```

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

i) Let *S* be the <time fractional seconds precision> of *SQLT*. Let ***SLIT*** be the canonical XML Schema literal denoting *S* in the lexical representation of XML Schema type **xs: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:

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

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

```
<xs: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:

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

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

```
<xs:simpleType>
  ANN
  <xs:restriction base="xs:dateTime">
    FACETP
  </xs:restriction>
</xs: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 the canonical XML Schema literal for *P* in the XML Schema type **`xs:integer`**. It is implementation-defined 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 the canonical XML Schema literal for *S* in the XML Schema type **`xs:integer`**. Let the XML text ***SECS*** be:

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

It is implementation-defined 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:

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

B) It is implementation-defined 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:

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

B) It is implementation-defined 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:

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

B) It is implementation-defined 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:

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

B) It is implementation-defined 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:

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

B) It is implementation-defined 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:

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

B) It is implementation-defined 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:

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

B) It is implementation-defined 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:

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

B) It is implementation-defined 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:

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

B) It is implementation-defined 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:

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

B) It is implementation-defined 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:

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

B) It is implementation-defined 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:

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

B) It is implementation-defined 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:

```
<xs:pattern value="-?PTSECS"/>
```

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

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

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

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

v) Case:

1) If *SQLT* is a year-month interval, then let ***DTYPE*** be `xs:yearMonthDuration`.

2) If *SQLT* is a day-time interval, then let ***DTYPE*** be `xs:dayTimeDuration`.

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

```
<xs:simpleType>
```



```
  ANN
  <xs:restriction base="DTYPE">
    FACETP
  </xs:restriction>
</xs:simpleType>
```

m) If *SQLT* is a domain, then:

i) Let *DT* be the data type of *SQLT*.

ii) The General Rules of Subclause 9.4, "Mapping an SQL data type to an XML Name", are applied with *DT* as *DATA TYPE*; let ***XMLN*** be the *XML NAME* returned from the application of those General Rules.

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:

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

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

```
<xs:simpleType>
  ANN
  <xs:restriction base="XMLN"/>
</xs: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) For *i* between 1 (one) and *N*, the General Rules of Subclause 9.4, "Mapping an SQL data type to an XML Name", are applied with FT_i as *DATA TYPE*; let $\boldsymbol{XMLMT_i}$ be the *XML NAME* returned from the application of those General Rules.

iii) For each *i* between 1 (one) and *N*, the General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with FN_i as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let $\boldsymbol{XMLFN_i}$ be the *XMLNAME* returned from the application of those General Rules.

iv) Let $\boldsymbol{FNLIT_i}$ 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:

```
<xs:annotation>
  <xs:appinfo>
```

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

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

```
<xs:complexType>
  ANN
  <xs:sequence>
    <xs:element name="XMLFN1" type="XMLMT1" XMLNULLS/>
    . . .
    <xs:element name="XMLFNN" type="XMLMTN" XMLNULLS/>
  </xs:sequence>
</xs:complexType>
```

o) If *SQLT* is a distinct type, then:

i) Let *ST* be the source type of *SQLT*.

ii) The General Rules of Subclause 9.4, "Mapping an SQL data type to an XML Name", are applied with *ST* as *DATA TYPE*; let ***XMLN*** be the *XML NAME* returned from the application of those General Rules.

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:

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

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

```
<xs:simpleType>
  ANN
  <xs:restriction base="XMLN"/>
</xs: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) The General Rules of Subclause 9.4, "Mapping an SQL data type to an XML Name", are applied with *ET* as *DATA TYPE*; let ***XMLN*** be the *XML NAME* returned from the application of those General Rules.

iii) Let ***MLIT*** be the canonical XML Schema literal for *M* in the XML Schema type **xs:integer**.

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

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

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

```
<xs:complexType>
  ANN
  <xs:sequence>
    <xs:element name="element" minOccurs="0"
                maxOccurs="MLIT" nillable="true"
                type="XMLN"/>
  </xs:sequence>
</xs:complexType>
```

q) If *SQLT* is a multiset type, then:

i) Let *ET* be the element type of *SQLT*.

ii) The General Rules of Subclause 9.4, "Mapping an SQL data type to an XML Name", are applied with *ET* as *DATA TYPE*; let ***XMLN*** be the *XML NAME* returned from the application of those General Rules.

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

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="MULTISET"
                    mappedElementType="XMLN"/>
  </xs:appinfo>
</xs:annotation>
```

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

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

```
<xs:annotation>
  <xs:appinfo>
    <sqlxml:sqltype kind="PREDEFINED"
                    name="XML"/>
  </xs:appinfo>
</xs:annotation>
```

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

```
<xs:complexType mixed="true">
  ANN
  <xs:sequence>
    <xs:any name="element" minOccurs="0" maxOccurs="unbounded"
            processContents="skip"/>
  </xs:sequence>
</xs:complexType>
```

8) ***XMLT*** is the result of this mapping.

Conformance Rules

None.

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

Subclause Signature

```
"Mapping an SQL data type to a named XML Schema data type" [General Rules] (
  Parameter: "SQLTYPE"
) Returns: "SCHEMA 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 *SQLTYPE* in an application of the General Rules of this Subclause. The result of the application of this Subclause is ***XSLCDT***, which is returned as *SCHEMA TYPE*.

   NOTE 58 — *D* is an SQL data type or domain.

2) The General Rules of Subclause 9.4, "Mapping an SQL data type to an XML Name", are applied with *D* as *DATA TYPE*; let ***XMLN*** be the *XML NAME* returned from the application of those General Rules.

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 *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 **xs: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:

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

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

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

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

a) Let *ENC* be the choice of whether to encode binary strings in base64 or in hex and let *NC* be the choice of whether to map null values to absent elements or elements that are marked with **xsi:nil="true"**.

b) The General Rules of Subclause 9.5, "Mapping SQL data types to XML Schema data types", are applied with *D* as *SQLTYPE*, *ENC* as *ENCODING*, and *NC* as *NULLS*; let *XMLT* be the *SCHEMA TYPE* returned from the application of those General Rules.

c) Case:

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

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

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

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

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

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

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

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

```
<xs:simpleType name="XMLN">
```



```
      <xs:restriction base="XMLT" />
    </xs:simpleType>
```

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

Conformance Rules

None.

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

Subclause Signature

```
"Mapping a collection of SQL data types to XML Schema data types" [General Rules] (
  Parameter: "NULLS",
  Parameter: "ENCODING",
  Parameter: "SQLTYPES"
) Returns: "SCHEMA TYPE"
```

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 *NULLS*, let *ENCODING* be the *ENCODING*, and let *C* be the *SQLTYPES* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XMLD*, which is returned as *SCHEMA TYPE*.

   NOTE 59 — *C* is a collection of SQL data types and domains.

2) *C* is augmented recursively as follows, until no more data types are added to *C*:

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

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

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

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

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

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

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

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

b) The General Rules of Subclause 9.4, "Mapping an SQL data type to an XML Name", are applied with D_i as *DATA TYPE*; let ***XMLN***$_i$ be the *XML NAME* returned from the application of those General Rules.

c) The General Rules of Subclause 9.5, "Mapping SQL data types to XML Schema data types", are applied with D_i as *SQLTYPE*, *ENCODING* as *ENCODING*, and *NULLS* as *NULLS*; let *XMLT*$_i$ be the *SCHEMA TYPE* returned from the application of those General Rules.

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***.

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

Conformance Rules

None.

9.8 Mapping values of SQL data types to values of XML Schema data types

Subclause Signature

```
"Mapping values of SQL data types to values of XML Schema data types" [General Rules]
(
  Parameter: "SQLVALUE",
  Parameter: "NULLS",
  Parameter: "ENCODING",
  Parameter: "CHARMAPPING"
) Returns: "XMLVALUE"
```

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 *SQLVALUE*, let *NL* be the *NULLS*, let *ENC* be the *ENCODING*, and let *CHARMAPPING* be the *CHARMAPPING* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XMLV*, which is returned as *XMLVALUE*.

   NOTE 60 — *NULLS* specifies whether to map null values to absent elements or to elements that are marked with `xsi:nil="true"`. *ENCODING* specifies whether to encode binary strings in base64 or hex. *CHARMAPPING* specifies 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*).

2) 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*.

3) The General Rules of Subclause 9.5, "Mapping SQL data types to XML Schema data types", are applied with *SQLT* as *SQLTYPE*, *ENC* as *ENCODING*, and *NL* as *NULLS*; let *XMLT* be the *SCHEMA TYPE* returned from the application of those General Rules.

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

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

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

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

7) Case:

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

i) Let *CS* be the character set of *SQLT*. Let ***XMLVRAW*** be the XML text that is the result of mapping *SQLV* to Unicode using the implementation-defined mapping of character strings of *CS* to Unicode.

ii) Case:

1) If the SQL-implementation supports Feature X211, "XML 1.1 support", then if any Unicode code point in ***XMLVRAW*** does not represent a valid XML 1.1 character, then an exception condition is raised: *SQL/XML mapping error — invalid XML character*.

2) Otherwise, if any Unicode code point in ***XMLVRAW*** does not represent a valid XML 1.0 character, then an exception condition is raised: *SQL/XML mapping error — invalid XML character*.

iii) 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) Case:

1) If the SECOND field of *CV* is greater than or equal to 60, then it is implementation-defined whether an implementation-defined value is assigned to *TEMP*, or whether to raise an exception condition: *data exception — datetime field overflow*.

2) Otherwise, 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 <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) Case:

1) If the SECOND field of *CV* is greater than or equal to 60, then it is implementation-defined whether an implementation-defined value is assigned to *TEMP*, or whether to raise an exception condition: *data exception — datetime field overflow*.

2) Otherwise, 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) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with FN_i as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let $\boldsymbol{XMLFN_i}$ be the *XMLNAME* returned from the application of those General Rules.

2) Case:

A) If FV_i is the null value and *NULLS* is "absent", then let $\boldsymbol{XMLE_i}$ be the zero-length string.

B) If FV_i is the null value and *NULLS* is "nil", then let $\boldsymbol{XMLE_i}$ be the XML element:

$\boldsymbol{<XMLFN_i\ xsi:nil="true"/>}$

C) Otherwise, let the XML text $\boldsymbol{XMLV_i}$ be the result of applying the mapping defined in this Subclause to FV_i. Let $\boldsymbol{XMLE_i}$ be the XML element:

$\boldsymbol{<XMLFN_i>XMLV_i</XMLFN_i>}$

iii) Let ***XMLV*** be:

$\boldsymbol{XMLE_1\ XMLE_2\ \ldots\ XMLE_N}$

j) If *SQLV* is an array value or a multiset value, then:

i) Let *N* be the number of elements in *SQLV*.

ii) Let *ET* be the element type of *SQLV*.

iii) For *i* between 1 (one) and *N*:

1) Let E_i be the value of the *i*-th element of *SQLV*. (If *SQLV* is a multiset value, then the ordering of the elements is implementation-dependent.)

2) Case:

A) If E_i is the null value, 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>*X* $_i$**</element>**

iv) Let ***XMLV*** be:

XMLE $_1$ ***XMLE*** $_2$ ... ***XMLE*** $_N$

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)
```

8) ***XMLV*** is the result of this mapping.

Conformance Rules

None.

9.9 Mapping an SQL table to XML Schema data types

Subclause Signature

```
"Mapping an SQL table to XML Schema data types" [General Rules] (
  Parameter: "TABLE",
  Parameter: "NULLS",
  Parameter: "TABLEFOREST",
  Parameter: "AUTHID"
) Returns: "SCHEMATYPES"
```

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*, let *NULLS* be the *NULLS*, let *TABLEFOREST* be the *TABLEFOREST*, and let *U* be the *AUTHID* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XMLTYPMAP*, which is returned as *SCHEMATYPES*.

   NOTE 61 — *NULLS* specifies 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"). *TABLEFOREST* specifies the choice of whether to map the table to a sequence of XML elements (*True*) or to map the table to an XML document with a single root element (*False*). *U* is 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 *n* be the number of XML visible columns of *T* for *U*.

   NOTE 62 — "XML visible column" is defined in Subclause 4.10.8, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

4) 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) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *CN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLCN*** be the *XMLNAME* returned from the application of those General Rules.

d) The General Rules of Subclause 9.4, "Mapping an SQL data type to an XML Name", are applied with *D* as *DATA TYPE*; let ***XMLCTN*** be the *XML NAME* returned from the application of those General Rules.

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) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *CSC* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLCSCN*** be the *XMLNAME* returned from the application of those General Rules.

4) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *CSS* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLCSSN*** be the *XMLNAME* returned from the application of those General Rules.

5) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *CSN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLCSN*** be the *XMLNAME* returned from the application of those General Rules.

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

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

8) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *COC* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLCOCN*** be the *XMLNAME* returned from the application of those General Rules. The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *COS* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLCOSN*** be the *XMLNAME* returned from the application of those General Rules. The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *CON* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLCON*** be the *XMLNAME* returned from the application of those General Rules.

9) It is implementation-defined whether ***COLANN*** is the zero-length string or

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

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

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

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

5) The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "TableType", *TC*, *TS*, and *TN* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLTYPEN*** be the *XMLNAME* returned from the application of those General Rules.

6) The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "RowType", *TC*, *TS*, and *TN* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLROWN*** be the *XMLNAME* returned from the application of those General Rules.

7) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *TC* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLCAT*** be the *XMLNAME* returned from the application of those General Rules. The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *TS* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLSN*** be the *XMLNAME* returned from the application of those General Rules. The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *TN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLTN*** be the *XMLNAME* returned from the application of those General Rules.

8) If *T* is a base table, then let ***TYPE*** be **BASE TABLE**; otherwise, let ***TYPE*** be **VIEWED TABLE**. It is implementation-defined whether ***SQLANN*** is the zero-length string or

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

9) Case:

a) If *TABLEFOREST* is *False*, then let ***XMLTYPMAP*** be:

```
<xs:complexType name="XMLROWN">
   <xs:sequence>
      XMLCE1
      ...
      XMLCEn
   </xs:sequence>
</xs:complexType>
<xs:complexType name="XMLTYPEN">
   SQLANN
   <xs:sequence>
      <xs:element name="row"
         type="XMLROWN"
         minOccurs="0"
         maxOccurs="unbounded" />
   </xs:sequence>
</xs:complexType>
```

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

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

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

Conformance Rules

None.

9.10 Mapping an SQL table to an XML element or a sequence of XML elements

Subclause Signature

```
"Mapping an SQL table to an XML element or a sequence of XML elements" [General Rules]
(
  Parameter: "TABLE",
  Parameter: "NULLS",
  Parameter: "TABLEFOREST",
  Parameter: "TARGETNS",
  Parameter: "ENCODING",
  Parameter: "SCHEMALOCATION",
  Parameter: "AUTHID"
) Returns: "XMLELEMS"
```

Function

Define the mapping of an SQL table to XML.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *T* be the *TABLE*, let *NULLS* be the *NULLS*, let *TABLEFOREST* be the *TABLEFOREST*, let *TARGETNS* be the *TARGETNS*, let *ENCODING* be the *ENCODING*, let *XSL* be the *SCHEMALOCATION*, and let *U* be the *AUTHID* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XMLTE*, which is returned as *XMLELEMS*.

   NOTE 63 — *NULLS* specifies the choice of whether to map null values to absent elements ("absent") or to elements that are marked with **`xsi:nil="true"`** ("nil"). *TABLEFOREST* specifies the choice of whether to map the table to a sequence of XML elements (*True*) or to an XML document with a single root element (*False*). ***TARGETNS*** specifies 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. *U* is the authorization identifier that is invoking this mapping. *ENCODING* specifies the choice of whether to encode binary strings in base64 or in hex. *SCHEMALOCATION* is the URI of the XML Schema describing the result of the data mapping, if any.

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

3) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *TN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLTN*** be the *XMLNAME* returned from the application of those General Rules.

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 64 — "XML visible column" is defined in Subclause 4.10.8, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

5) Case:

   a) If *XSL* is not the zero-length string, then:

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

         xsi:noNamespaceSchemaLocation="*XSL*"

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

         xsi:schemaLocation="*TARGETNS XSL*"

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

6) Let ***XSINS*** be the value of the XML namespace URI provided for the XML namespace prefix **xsi** in Table 2, "XML namespace prefixes and their URIs".

   Case:

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

   b) Otherwise, let ***XSI*** be

      xmlns:xsi="*XSINS*"

7) 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) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with CN_j as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let $\mathbf{XMLCN}_j$ be the *XMLNAME* returned from the application of those General Rules.

      iv) Let V_j be the value of C_j for R_i.

      v) Case:

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

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

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

         3) Otherwise:

            A) The General Rules of Subclause 9.8, "Mapping values of SQL data types to values of XML Schema data types", are applied with V_j as *SQLVALUE*, *ENCODING* as

ENCODING, *NULLS* as *NULLS*, and *True* as *CHARMAPPING*; let $XMLV_j$ be the *XMLVALUE* returned from the application of those General Rules.

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

<XMLCN$_j$***>XMLV***$_j$***</XMLCN***$_j$***>***

c) Case:

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

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

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

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

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

```
<XMLTN XSI xmlns="TARGETNS" XSLA>
   XMLC1
   ...
   XMLCm
</XMLTN>
```

8) Case:

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

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

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

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

c) If *TABLEFOREST* is *True*, then let ***XMLTE*** be:

XMLR$_1$
...
XMLR$_n$

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

Conformance Rules

None.

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

Subclause Signature

```
"Mapping an SQL table to XML and an XML Schema document" [General Rules] (
  Parameter: "TABLE",
  Parameter: "NULLS",
  Parameter: "TABLEFOREST",
  Parameter: "TARGETNS",
  Parameter: "ENCODING",
  Parameter: "RETURN",
  Parameter: "AUTHID"
) Returns: "RESULT"
```

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 a sequence of XML elements.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *T* be the *TABLE*, let *NULLS* be the *NULLS*, let *TABLEFOREST* be the *TABLEFOREST*, let ***TARGETNS*** be the *TARGETNS*, let *ENCODING* be the *ENCODING*, let *RETURN* be the *RETURN*, and let *U* be the *AUTHID* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XRET*, which is returned as *RESULT*.

   NOTE 65 — *NULLS* specifies the choice of whether to map null values to absent elements ("absent") or to elements that are marked with **`xsi:nil="true"`** ("nil"). *TABLEFOREST* specifies the choice of whether to map the table to a sequence of XML elements (*True*) or to an XML document with a single XML element (*False*). ***TARGETNS*** is 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. *ENCODING* specifies the choice of whether binary strings are to be encoded in base64 or in hex. *RETURN* specifies the choice of whether the mapping results in an XML representation of the data of *T*, an XML Schema document that describes the XML representation of the data of *T*, or both. *U* is 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) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *TN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLTN*** be the *XMLNAME* returned from the application of those General Rules.

4) 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*.

5) The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "TableType", *TC*, *TS*, and *TN* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLTYPEN*** be the *XML-NAME* returned from the application of those General Rules.

6) The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "RowType", *TC*, *TS*, and *TN* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLROWN*** be the *XML NAME* returned from the application of those General Rules.

7) Let *CT* be the XML visible columns of *T* for *U*.

8) The General Rules of Subclause 9.7, "Mapping a collection of SQL data types to XML Schema data types", are applied with the data types and domains of the columns of *CT* as *SQLTYPES*, *ENCODING* as *ENCODING*, and *NULLS* as *NULLS*; let *XSCT* be the *SCHEMA TYPE* returned from the application of those General Rules.

NOTE 66 — "XML visible column" is defined in Subclause 4.10.8, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

9) The General Rules of Subclause 9.9, "Mapping an SQL table to XML Schema data types", are applied with *T* as *TABLE*, *NULLS* as *NULLS*, *TABLEFOREST* as *TABLEFOREST*, and *U* as *AUTHID*; let *XST* be the *SCHEMATYPES* returned from the application of those General Rules.

10) 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.

11) The character 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 *CHARENC* be the name of the character encoding, and let ***ENC*** be

```
encoding = 'CHARENC'
```

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

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

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

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

13) If *RETURN* is not "data only", then:

a) Let ***SQLXMLNS*** be the value of the XML namespace URI 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 URI provided for the XML namespace prefix **xs** 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

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

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

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

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


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

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

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

f) Let **XSR** be:

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

14) Case:

a) If *RETURN* is not "data only", then let **XSL** be the URI that identifies **XSR**.

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

15) If *RETURN* is not "metadata only", then:

a) The General Rules of Subclause 9.10, "Mapping an SQL table to an XML element or a sequence of XML elements", are applied with *T* as *TABLE*, *NULLS* as *NULLS*, *TABLEFOREST* as *TABLEFOREST*, *TARGETNS* as *TARGETNS*, *ENCODING* as *ENCODING*, *XSL* as *SCHEMALOCATION*, and *U* as *AUTHID*; let *XDROWS* be the *XMLELEMS* returned from the application of those General Rules.

b) Case:

i) If *TABLEFOREST* is *False*, then **XD** has the following contents:

```
XMLDECL
   XDROWS
```

ii) If *TABLEFOREST* is *True*, then **XD** is **XDROWS**.

c) 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 )
```

16) Case:

a) If *RETURN* is "metadata only", then let *XRET* be ***XSR*** (the character large object value containing the XML Schema document that describes the XML representation of the data of T).

b) If *RETURN* is "data only", then let *XRET* be ***XDR*** (the character large object value containing the XML document or forest of XML elements that is the XML representation of the data of T).

c) Otherwise, let *XRET* be:

```
XSR || '&&' || XDR
```

Conformance Rules

1) Without Feature X040, "Basic table mapping", a conforming application shall not invoke this Subclause of this part of this International Standard.

2) Without Feature X041, "Basic table mapping: null 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.

3) 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"`.

4) 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*.

5) 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*.

6) 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.

7) Without Feature X046, "Basic table mapping: data mapping", a conforming application shall not invoke this Subclause of this part of this International Standard with *RETURN* not set to "metadata only".

8) Without Feature X047, "Basic table mapping: metadata mapping", a conforming application shall not invoke this Subclause of this part of this International Standard with *RETURN* not set to "data only".

9) 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.

10) 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.

9.12 Mapping an SQL schema to XML Schema data types

Subclause Signature

```
"Mapping an SQL schema to XML Schema data types" [General Rules] (
  Parameter: "SCHEMA",
  Parameter: "NULLS",
  Parameter: "TABLEFOREST",
  Parameter: "AUTHID"
) Returns: "SCHEMATYPES"
```

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*, let *NULLS* be the *NULLS*, let *TABLEFOREST* be the *TABLEFOREST*, and let *U* be the *AUTHID* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XMLSCHEMAT*, which is returned as *SCHEMATYPES*.

   NOTE 67 — *NULLS* specifies the choice of whether to map null values to absent elements ("absent") or to elements that are marked with `xsi:nil="true"` ("nil"). *TABLEFOREST* specifies the choice of whether to map each table of the schema to a sequence of XML elements, one for each row of the table (*True*) or to map the table to a single XML element that contains an element for each row of the table (*False*). *U* is 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) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *SN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLSN*** be the *XMLNAME* returned from the application of those General Rules.

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

   NOTE 68 — "XML visible table" is defined in Subclause 4.10.8, "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) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *TN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLTN*** be the *XMLNAME* returned from the application of those General Rules.

d) The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "TableType", *SC*, *SN*, and *TN* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLTTN*** be the *XMLNAME* returned from the application of those General Rules.

e) The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "RowType", *SC*, *SN*, and *TN* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLROWN*** be the *XMLNAME* returned from the application of those General Rules.

f) The General Rules of Subclause 9.9, "Mapping an SQL table to XML Schema data types", are applied with T_i as *TABLE*, *NULLS* as *NULLS*, *TABLEFOREST* as *TABLEFOREST*, and *U* as *AUTHID*; let $\textbf{\textit{XMLTTYPE}}_i$ be the *SCHEMATYPES* returned from the application of those General Rules

g) Case:

i) If *TABLEFOREST* is *False*, then let $\textbf{\textit{XMLTE}}_i$ be

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

ii) If *TABLEFOREST* is *True*, then let $\textbf{\textit{XMLTE}}_i$ be

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

6) The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "SchemaType", *SC*, and *SN* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLTYPEN*** be the *XMLNAME* returned from the application of those General Rules.

7) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *SC* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLSC*** be the *XMLNAME* returned from the application of those General Rules.

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

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

9) Case:

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

```
XMLTTYPE1
...
XMLTTYPEn
```

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

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

```
XMLTTYPE1
...
XMLTTYPEn
<xs:complexType name="XMLTYPEN">
   SQLANN
   <xs:sequence>
      XMLTE1
      ...
      XMLTEn
   </xs:sequence>
</xs:complexType>
```

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

Conformance Rules

None.

9.13 Mapping an SQL schema to an XML element

Subclause Signature

```
"Mapping an SQL schema to an XML element" [General Rules] (
  Parameter: "SCHEMA",
  Parameter: "NULLS",
  Parameter: "TABLEFOREST",
  Parameter: "TARGETNS",
  Parameter: "ENCODING",
  Parameter: "SCHEMALOCATION",
  Parameter: "AUTHID"
) Returns: "XMLELEMENT"
```

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*, let *NULLS* be the *NULLS*, let *TABLEFOREST* be the *TABLEFOREST*, let ***TARGETNS*** be the *TARGETNS*, let *ENCODING* be the *ENCODING*, let *XSL* be the *SCHEMALOCATION*, and let *U* be the *AUTHID* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XMLSE*, which is returned as *XMLELEMENT*.

   NOTE 69 — *NULLS* specifies the choice of whether to map null values to absent elements ("absent") or to elements that are marked with `xsi:nil="true"` ("nil"). *TABLEFOREST* specifies the choice of whether to map each table of the schema to a sequence of XML elements, one for each row of the table (*True*), or to a single XML element that contains an element for each row of the table (*False*). ***TARGETNS*** is 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. *U* is the authorization identifier that is invoking this mapping. *ENCODING* specifies the choice of whether to encode binary strings in base64 or in hex. of the data mapping, if any.

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

3) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *SN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLSN*** be the *XMLNAME* returned from the application of those General Rules.

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

   NOTE 70 — "XML visible table " is defined in Subclause 4.10.8, "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) The General Rules of Subclause 9.10, "Mapping an SQL table to an XML element or a sequence of XML elements", are applied with T_i as *TABLE*, *NULLS* as *NULLS*, *TABLEFOREST* as *TABLEFOREST*, *TARGETNS* as *TARGETNS*, *ENCODING* as *ENCODING*, *XSL* as *SCHEMALOCATION*, and *U* as *AUTHID*; let ***XMLT***$_i$ be the *XMLELEMS* returned from the application of those General Rules.

6) Case:

   a) If *XSL* is not the zero-length string, then:

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

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

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

         ```
         xsi:schemaLocation="TARGETNS XSI"
         ```

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

7) Let *XSINS* be the value of the XML namespace URI provided for the XML namespace prefix `xsi` in Table 2, "XML namespace prefixes and their URIs".

   Case:

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

   b) Otherwise, let ***XSI*** be:

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

8) Case:

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

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

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

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

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

Conformance Rules

None.

9.14 Mapping an SQL schema to an XML document and an XML Schema document

Subclause Signature

```
"Mapping an SQL schema to an XML document and an XML Schema document" [General Rules]
(
  Parameter: "SCHEMA",
  Parameter: "NULLS",
  Parameter: "TABLEFOREST",
  Parameter: "TARGETNS",
  Parameter: "ENCODING",
  Parameter: "RETURN",
  Parameter: "AUTHID"
) Returns: "RESULT"
```

Function

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

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *S* be the *SCHEMA*, let *NULLS* be the *NULLS*, let *TABLEFOREST* be the *TABLEFOREST*, let ***TARGETNS*** be the *TARGETNS*, let *ENCODING* be the *ENCODING*, let *RETURN* be the *RETURN*, and let *U* be the *AUTHID* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XRET*, which is returned as *RESULT*.

   NOTE 71 — *NULLS* specifies the choice of whether to map null values to absent elements ("absent") or to elements that are marked with **`xsi:nil="true"`** ("nil"). *TABLEFOREST* specifies the choice of whether to map each table of the schema to a sequence of XML elements, one for each row of the table (*True*), or to a single XML element that contains an element for each row of the table (*False*). ***TARGETNS*** is 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. *ENCODING* specifies the choice of whether binary strings are to be encoded in base64 or in hex. *RETURN* specifies the choice of whether the mapping results in an XML representation of the data of *T*, an XML Schema document that describes the XML representation of the data of *T*, or both. *U* is the authorization identifier that is invoking this mapping.

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

3) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *SN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLSN*** be the *XMLNAME* returned from the application of those General Rules.

4) 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 72 — "visible column" is defined in Subclause 4.10.8, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

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

6) The General Rules of Subclause 9.7, "Mapping a collection of SQL data types to XML Schema data types", are applied with the data types and domains of the columns of *CT* as *SQLTYPES*, *ENCODING* as *ENCODING*, and *NULLS* as *NULLS*; let *XSCT* be the *SCHEMA TYPE* returned from the application of those General Rules.

7) The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "SchemaType", *SC* and *SN* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLTYPEN*** be the *XMLNAME* returned from the application of those General Rules.

8) The General Rules of Subclause 9.12, "Mapping an SQL schema to XML Schema data types", are applied with *S* as *SCHEMA*, *NULLS* as *NULLS*, *TABLEFOREST* as *TABLEFOREST*, and *U* as *AUTHID*; let *XST* be the *SCHEMATYPES* returned from the application of those General Rules.

9) 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.

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

```
encoding = 'CHARENC'
```

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

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

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

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

12) If *RETURN* is not "data only", then:

a) Let ***SQLXMLNS*** be the value of the XML namespace URI 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 URI provided for the XML namespace prefix **xs** 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

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

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

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

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

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

13) Case:

a) If *RETURN* is not "data only", then let ***XSL*** be the URI that identifies ***XSR***.

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

14) If *RETURN* is not "metadata only", then:

a) The General Rules of Subclause 9.13, "Mapping an SQL schema to an XML element", are applied with *S* as *SCHEMA*, *NULLS* as *NULLS*, *TABLEFOREST* as *TABLEFOREST*, *TARGETNS* as *TARGETNS*, *XSL* as *SCHEMALOCATION*, *U* as *AUTHID*, and *ENCODING* as *ENCODING*; let *XDSCHEMA* be the *XMLELEMENT* returned from the application of those General Rules.

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

```
XMLDECL
   XDSCHEMA
```

c) Let *XDR* be:

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

15) Case:

a) If *RETURN* is "metadata only", then let *XRET* be ***XSR*** (the character large object value containing the XML Schema document that describes the XML representation of the data of *S*).

b) If *RETURN* is "data only", then let *XRET* be ***XDR*** (the character large object value containing the XML document or forest of XML elements that is the XML representation of the data of *S*).

c) Otherwise, let *XRET* be:

```
XSR || '&&' || XDR
```

Conformance Rules

1) Without Feature X050, "Advanced table mapping", a conforming application shall not invoke this Subclause of this part of this International Standard.

2) Without Feature X051, "Advanced table mapping: null 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.

3) 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"`**.

4) 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*.

5) 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*.

6) 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.

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

8) Without Feature X057, “Advanced table mapping: metadata mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with *RETURN* not set to “data only”.

9) 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.

10) 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.

9.15 Mapping an SQL catalog to XML Schema data types

Subclause Signature

```
"Mapping an SQL catalog to XML Schema data types" [General Rules] (
  Parameter: "CATALOG",
  Parameter: "NULLS",
  Parameter: "TABLEFOREST",
  Parameter: "AUTHID"
) Returns: "SCHEMATYPES"
```

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*, let *NULLS* be the *NULLS*, let *TABLEFOREST* be the *TABLEFOREST*, and let *U* be the *AUTHID* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XMLCATT*, which is returned as *SCHEMATYPES*.

   NOTE 73 — *NULLS* specifies the choice of whether to map null values to absent elements ("absent") or to elements that are marked with `xsi:nil="true"` ("nil"). *TABLEFOREST* specifies the choice of whether to map each table of the schema to a sequence of XML elements, one for each row of the table (*True*) or to map the table to a single XML element that contains an element for each row of the table (*False*). *U* is the authorization identifier that is invoking this mapping.

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

3) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *CN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLCN*** be the *XMLNAME* returned from the application of those General Rules.

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

   NOTE 74 — "XML visible schema" is defined in Subclause 4.10.8, "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) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *SN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLSN*** be the *XMLNAME* returned from the application of those General Rules.

d) The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "SchemaType", *CN* and *SN* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLSTN*** be the *XMLNAME* returned from the application of those General Rules.

e) The General Rules of Subclause 9.12, "Mapping an SQL schema to XML Schema data types", are applied with S_i as *SCHEMA*, *NULLS* as *NULLS*, *TABLEFOREST* as *TABLEFOREST*, and *U* as *AUTHID*; let $\textbf{\textit{XMLSTYPE}}_i$ be the *SCHEMATYPES* returned from the application of those General Rules

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

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

6) The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "CatalogType" and *CN* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLTYPEN*** be the *XMLNAME* returned from the application of those General Rules.

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

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

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

```
XMLSTYPE1
...
XMLSTYPEn
<xs:complexType name="XMLTYPEN">
   SQLANN
   <xs:all>
      XMLSE1
      ...
      XMLSEn
   </xs:all>
</xs:complexType>
```

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

Conformance Rules

None.

9.16 Mapping an SQL catalog to an XML element

Subclause Signature

```
"Mapping an SQL catalog to an XML element" [General Rules] (
  Parameter: "CATALOG",
  Parameter: "NULLS",
  Parameter: "TABLEFOREST",
  Parameter: "TARGETNS",
  Parameter: "ENCODING",
  Parameter: "SCHEMALOCATION",
  Parameter: "AUTHID"
) Returns: "XMLELEMENT"
```

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*, let *NULLS* be the *NULLS*, let *TABLEFOREST* be the *TABLEFOREST*, let ***TARGETNS*** be the *TARGETNS*, let *ENCODING* be the *ENCODING*, let *XSL* be the *SCHEMALOCATION*, and let *U* be the *AUTHID* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XMLSE*, which is returned as *XMLELEMENT*.

   NOTE 75 — *NULLS* specifies the choice of whether to map null values to absent elements ("absent") or to elements that are marked with **`xsi:nil="true"`** ("nil"). *TABLEFOREST* specifies the choice of whether to map each table to a sequence of XML elements, one for each row of the table (*True*), or to a single XML element that contains an element for each row of the table (*False*). ***TARGETNS*** is 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. *U* is the authorization identifier that is invoking this mapping. *ENCODING* specifies the choice of whether to encode binary strings in base64 or in hex. ***SCHEMALOCATION*** is the URI of the XML Schema describing the result of the data mapping, if any.

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

3) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *CN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLCN*** be the *XMLNAME* returned from the application of those General Rules.

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

   NOTE 76 — "XML visible schema" is defined in Subclause 4.10.8, "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) The General Rules of Subclause 9.13, "Mapping an SQL schema to an XML element", are applied with S_i as *SCHEMA*, *NULLS* as *NULLS*, *TABLEFOREST* as *TABLEFOREST*, *TARGETNS* as *TARGETNS*, *ENCODING* as *ENCODING*, *XSL* as *SCHEMALOCATION*, and *U* as *AUTHID*; let $\textbf{\textit{XMLS}}_i$ be the *XMLELEMENT* returned from the application of those General Rules.

6) Case:

   a) If *XSL* is not the zero-length string, then:

      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.

7) Let ***XSINS*** be the value of the XML namespace URI provided for the XML namespace prefix `xsi` in Table 2, "XML namespace prefixes and their URIs".

   Case:

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

   b) Otherwise, let ***XSI*** be:

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

8) Case:

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

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

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

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

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

Conformance Rules

None.

9.17 Mapping an SQL catalog to an XML document and an XML Schema document

Subclause Signature

```
"Mapping an SQL catalog to an XML document and an XML Schema document" [General Rules]
(
  Parameter: "CATALOG",
  Parameter: "NULLS",
  Parameter: "TABLEFOREST",
  Parameter: "TARGETNS",
  Parameter: "ENCODING",
  Parameter: "RETURN",
  Parameter: "AUTHID"
) Returns: "RESULT"
```

Function

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

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *C* be the *CATALOG*, let *NULLS* be the *NULLS*, let *TABLEFOREST* be the *TABLEFOREST*, let ***TARGETNS*** be the *TARGETNS*, let *ENCODING* be the *ENCODING*, let *RETURN* be the *RETURN*, and let *U* be the *AUTHID* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *XRET*, which is returned as *RESULT*.

   NOTE 77 — *NULLS* specifies the choice of whether to map null values to absent elements ("absent") or to elements that are marked with **xsi:nil="true"** ("nil"). *TABLEFOREST* specifies the choice of whether to map each table of the catalog to a sequence of XML elements, one for each row of the table (*True*), or to a single XML element that contains one element for each row of the table (*False*). ***TARGETNS*** is 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. *ENCODING* specifies the choice of whether binary strings are to be encoded in base64 or in hex. *RETURN* specifies the choice of whether the mapping results in an XML representation of the data of *T*, an XML Schema document that describes the XML representation of the data of *T*, or both. *U* is the authorization identifier that is invoking this mapping.

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

3) The General Rules of Subclause 9.1, "Mapping SQL <identifier>s to XML Names", are applied with *CN* as *IDENT* and *fully escaped* as *ESCAPE VARIANT*; let ***XMLCN*** be the *XMLNAME* returned from the application of those General Rules.

4) 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 78 — "visible column" is defined in Subclause 4.10.8, "Visibility of columns, tables, and schemas in mappings from SQL to XML".

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

6) The General Rules of Subclause 9.7, "Mapping a collection of SQL data types to XML Schema data types", are applied with the data types and domains of the columns of *CT* as *SQLTYPES*, *ENCODING* as *ENCODING*, and *NULLS* as *NULLS*; let *XSCT* be the *SCHEMA TYPE* returned from the application of those General Rules.

7) The General Rules of Subclause 9.2, "Mapping a multi-part SQL name to an XML Name", are applied with "CatalogType" and *CN* as *SEQUENCE OF SQL IDENTIFIERS*; let ***XMLTYPEN*** be the *XMLNAME* returned from the application of those General Rules.

8) The General Rules of Subclause 9.15, "Mapping an SQL catalog to XML Schema data types", are applied with *C* as *CATALOG*, *NULLS* as *NULLS*, *TABLEFOREST* as *TABLEFOREST*, and *U* as *AUTHID*; let *XST* be the *SCHEMATYPES* returned from the application of those General Rules.

9) 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.

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

```
encoding = 'CHARENC'
```

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

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

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

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

12) If *RETURN* is not "data only", then:

   a) Let ***SQLXMLNS*** be the value of the XML namespace URI 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 URI provided for the XML namespace prefix **xs** 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

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

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

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

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

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

13) Case:

a) If *RETURN* is not "data only", then let ***XSL*** be the URI that identifies ***XSR***.

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

14) If *RETURN* is not "metadata only", then:

a) The General Rules of Subclause 9.16, "Mapping an SQL catalog to an XML element", are applied with *C* as *CATALOG*, *NULLS* as *NULLS*, *TABLEFOREST* as *TABLEFOREST*, *TARGETNS* as *TARGETNS*, *ENCODING* as *ENCODING*, *XSL* as *SCHEMALOCATION*, and *U* as *AUTHID*; let *XDCATALOG* be the *XMLELEMENT* returned from the application of those General Rules.

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

```
XMLDECL
   XDCATALOG
```

c) Let *XDR* be:

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

15) Case:

a) If *RETURN* is "metadata only", then let *XRET* be ***XSR*** (the character large object value containing the XML Schema document that describes the XML representation of the data of *C*).

b) If *RETURN* is "data only", then let *XRET* be ***XDR*** (the character large object value containing the XML document or forest of XML elements that is the XML representation of the data of *C*).

c) Otherwise, let *XRET* be:

```
XSR || '&&' || XDR
```

Conformance Rules

1) Without Feature X050, "Advanced table mapping", a conforming application shall not invoke this Subclause of this part of this International Standard.

2) Without Feature X051, "Advanced table mapping: null 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.

3) 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"`**.

4) 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*.

5) 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*.

6) 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.

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

8) Without Feature X057, “Advanced table mapping: metadata mapping”, a conforming application shall not invoke this Subclause of this part of this International Standard with *RETURN* not set to “data only”.

9) 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.

10) 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.

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) Insert this SR If *TD* is an XML type, then

   Case:

   a) If *TD* is either XML(DOCUMENT(UNTYPED)) or XML(CONTENT(UNTYPED)), then *SD* shall be either XML(DOCUMENT(UNTYPED)) or XML(CONTENT(UNTYPED)).

   b) If *TD* is either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)), then

      Case:

      i) If the type descriptor of *TD* includes the XML namespace URI *N* and the XML NCName *EN* of a global element declaration schema component, then *SD* shall be either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)), the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *TD* shall be identical to the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *SD*, and the type descriptor of *SD* shall include an XML namespace URI that is identical to *N*, as defined by [Namespaces] and an XML NCName that is equivalent to *EN*.

      ii) If the type descriptor of *TD* includes the XML namespace URI *N*, then *SD* shall be either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)), the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *TD* shall be identical to the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *SD*, and the type descriptor of *SD* shall include an XML namespace URI that is identical to *N*, as defined by [Namespaces].

      iii) Otherwise, *SD* shall be either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) and the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *TD* shall be identical to the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *SD*.

c) Otherwise, *SD* shall be an XML type.

Access Rules

No additional Access Rules.

General Rules

1) Augment GR 7) If the declared type of *T* is an XML type, then:

   a) Case:

      i) If the declared type of *T* is XML(DOCUMENT(UNTYPED)), XML(DOCUMENT(ANY)), or XML(DOCUMENT(XMLSCHEMA)), and *V* is not an XQuery sequence of length 1 (one) whose sole XQuery item is an XQuery document node whose **children** property contains exactly one XQuery element node, zero or more XQuery comment nodes, and zero or more XQuery processing instruction nodes, then an exception condition is raised: *data exception — not an XML document*.

      ii) If the declared type of *T* is XML(CONTENT(UNTYPED)), XML(CONTENT(ANY)), or XML(CONTENT(XMLSCHEMA)), and *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) The value of *T* is set to *V*.

Conformance Rules

No additional Conformance Rules.

10.2 Store assignment

Subclause Signature

```
"Store assignment" [General Rules] (
  Parameter: "TARGET",
  Parameter: "VALUE",
  Parameter: "PASSING"
)
```

This Subclause modifies Subclause 9.2, "Store assignment", in ISO/IEC 9075-2.

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) Insert this SR If *TD* is an XML type, then

   Case:

   a) If *TD* is either XML(DOCUMENT(UNTYPED)) or XML(CONTENT(UNTYPED)), then *SD* shall be either XML(DOCUMENT(UNTYPED)) or XML(CONTENT(UNTYPED)).

   b) If *TD* is either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)), then

      Case:

      i) If the type descriptor of *TD* includes the XML namespace URI *N* and the XML NCName *EN* of a global element declaration schema component, then *SD* shall be either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)), the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *TD* shall be identical to the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *SD*, and the type descriptor of *SD* shall include an XML namespace URI that is identical to *N*, as defined by [Namespaces], and an XML NCName that is equivalent to *EN*.

      ii) If the type descriptor of *TD* includes the XML namespace URI *N*, then *SD* shall be either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)), the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *TD* shall be identical to the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *SD*, and the type descriptor of *SD* shall include an XML namespace URI that is identical to *N*, as defined by [Namespaces].

      iii) Otherwise, *SD* shall be either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) and the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *TD* shall be identical to the XML Schema identified by the registered XML Schema descriptor included in the type descriptor of *SD*.

   c) Otherwise, *SD* shall be an XML type.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR Let *T* be the *TARGET*, let *V* be the *VALUE*, and let *P* be the *PASSING* in an application of the General Rules of this Subclause.

2) Augment GR 3)b) If the declared type of *T* is the XML type, then:

   a) Case:

      i) If the declared type of *T* is XML(DOCUMENT(UNTYPED)), XML(DOCUMENT(ANY)), or XML(DOCUMENT(XMLSCHEMA)), and *V* is not an XQuery sequence of length 1 (one) whose sole XQuery item is an XQuery document node whose **children** property contains exactly one XQuery element node, zero or more XQuery comment nodes, and zero or more XQuery processing instruction nodes, then an exception condition is raised: *data exception — not an XML document*.

      ii) If the declared type of *T* is XML(CONTENT(UNTYPED)), XML(CONTENT(ANY)), or XML(CONTENT(XMLSCHEMA)), and *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) Case:

      i) If *T* is a <column reference> or a <target array element specification> whose <target array reference> is a <column reference>, or *P* is BY VALUE, then the General Rules of Subclause 10.18, "Constructing a copy of an XML value", are applied with ***V*** as *VALUE*; let ***V1*** be the *COPY* returned from the application of those General Rules.

      ii) Otherwise, let *V1* be *V*.

   c) The value of *T* is set to *V1*.

Conformance Rules

No additional Conformance Rules.

10.3 Result of data type combinations

This Subclause modifies Subclause 9.5, "Result of data type combinations", in ISO/IEC 9075-2.

Function

Specify the declared type of the result of certain combinations of values of compatible data types, such as <case expression>s, <collection value expression>s, or a column in the result of a <query expression>.

Syntax Rules

1) Insert after SR 3)f) If any data type in *DTS* is an XML type, then:

   a) Each data type in *DTS* shall be an XML type.

   b) Case:

      i) If any of the data types in *DTS* is XML(SEQUENCE), then the declared type of the result is XML(SEQUENCE).

      ii) Otherwise:

         1) Let *N* be the number of data types in *DTS*. Let T_i, 1 (one) ≤ *i* ≤ *N*, be the *i*-th data type in *DTS*.

         2) Case:

            A) If *N* = 1 (one), then the declared type of the result is T_1.

            B) Otherwise,

               Case:

               I) If T_i, for all *i*, 1 (one) ≤ *i* ≤ *N*, is XML(DOCUMENT(UNTYPED)), then the declared type of the result is XML(DOCUMENT(UNTYPED)).

               II) If T_i, for all *i*, 1 (one) ≤ *i* ≤ *N*, is either XML(DOCUMENT(UNTYPED)) or XML(CONTENT(UNTYPED)), then the declared type of the result is XML(CONTENT(UNTYPED)).

               III) If there exists a registered XML Schema *S* and a global element declaration schema component ***E*** such that, for all *i*, 1 (one) ≤ *i* ≤ *N*, T_i is XML(DOCUMENT(XMLSCHEMA)) and the XML type descriptor of T_i includes the registered XML Schema descriptor of *S* and an XML namespace URI ***NS*** that is identical to the XML namespace URI of ***E***, as defined by [Namespaces], and an XML NCName ***EN*** that is equivalent to the XML NCName of ***E***, then the declared type of the result is XML(DOCUMENT(XMLSCHEMA)) whose XML type descriptor includes the registered XML Schema descriptor of *S*, the XML namespace URI ***NS***, and the XML NCName ***EN***.

               IV) If there exists a registered XML Schema *S* and an XML namespace ***NS*** such that, for all *i*, 1 (one) ≤ *i* ≤ *N*, T_i is XML(DOCUMENT(XMLSCHEMA)) and

the XML type descriptor of T_i includes the registered XML Schema descriptor of *S* and an XML namespace URI that is identical to ***NS***, as defined by [Namespaces], then the declared type of the result is XML(DOCUMENT(XMLSCHEMA)) whose XML type descriptor includes the registered XML Schema descriptor of *S* and the XML namespace URI ***NS***.

V) If there exists a registered XML Schema *S* such that, for all *i*, 1 (one) ≤ *i* ≤ *N*, T_i is XML(DOCUMENT(XMLSCHEMA)) and the XML type descriptor of T_i includes the registered XML Schema descriptor of *S*, then the declared type of the result is XML(DOCUMENT(XMLSCHEMA)) whose XML type descriptor includes the registered XML Schema descriptor of *S*.

VI) If there exists a registered XML Schema *S* and a global element declaration schema component ***E*** such that, for all *i*, 1 (one) ≤ *i* ≤ *N*, T_i is either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) and the XML type descriptor of T_i includes the registered XML Schema descriptor of *S* and an XML namespace URI ***NS*** that is identical to the XML namespace URI of ***E***, as defined by [Namespaces], and an XML NCName ***EN*** that is equivalent to the XML NCName of ***E***, then the declared type of the result is XML(CONTENT(XMLSCHEMA)) whose XML type descriptor includes the registered XML Schema descriptor of *S*, the XML namespace URI ***NS***, and the XML NCName ***EN***.

VII) If there exists a registered XML Schema *S* and an XML namespace ***NS*** such that, for all *i*, 1 (one) ≤ *i* ≤ *N*, T_i is either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) and the XML type descriptor of T_i includes the registered XML Schema descriptor of *S* and an XML namespace URI that is identical to ***NS***, as defined by [Namespaces], then the declared type of the result is XML(CONTENT(XMLSCHEMA)) whose XML type descriptor includes the registered XML Schema descriptor of *S* and the XML namespace URI ***NS***.

VIII) If there exists a registered XML Schema *S* such that, for all *i*, 1 (one) ≤ *i* ≤ *N*, T_i is either XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)) and the XML type descriptor of T_i includes the registered XML Schema descriptor of *S*, then the declared type of the result is XML(CONTENT(XMLSCHEMA)) whose XML type descriptor includes the registered XML Schema descriptor of *S*.

IX) Otherwise, the declared type of the result is XML(CONTENT(ANY)).

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.4 Type precedence list determination

This Subclause modifies Subclause 9.7, "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.

10.5 Type name determination

This Subclause modifies Subclause 9.9, "Type name determination", in ISO/IEC 9075-2.

Function

Determine an <identifier> given the name of a predefined or collection 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.

10.6 Determination of identical values

This Subclause modifies Subclause 9.10, "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

Case:

a) If *V1* and *V2* are both XQuery sequences, then *V1* and *V2* are are identical if they have the same number of XQuery items and their corresponding XQuery items are identical.

b) If *V1* and *V2* are both XQuery atomic values, then *V1* and *V2* 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 ***YM1*** equals ***YM2*** and ***DT1*** equals ***DT2*** where ***YM1*** and ***YM2*** are XQuery atomic values of type **xs:yearMonthDuration** whose year and month components equal the year and month components of ***V1*** and ***V2***, respectively, and ***DT1*** and ***DT2*** are XQuery atomic values of type **xs:dayTimeDuration** whose day, hour, minute, and second components equal the day, hour, minute, and second components of ***V1*** and ***V2***, respectively.

ii) Otherwise, the result of comparing ***V1*** and ***V2*** using the XQuery "eq" operation is *True*.

c) If *V1* and *V2* are both XQuery nodes, then *V1* and *V2* are identical if the XQuery node identity of ***V1*** is the same as the XQuery node identity of ***V2***.

NOTE 79 — This definition is equivalent to the XQuery expression "***V1 is V2***".

d) Otherwise, *V1* and *V2* are not identical.

Conformance Rules

No additional Conformance Rules.

10.7 Determination of equivalent XML values

Function

Determine whether two instances of XML values are equivalent.

NOTE 80 — This Subclause is implicitly invoked whenever the word *equivalent* is used with respect to XML values.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *V1* and *V2* be two XML values specified in an application of this Subclause.

2) Whether *V1* and *V2* are equivalent or not equivalent is defined as follows:

NOTE 81 — This definition involves the simultaneous recursive definition of equivalent XQuery items and identical properties of XQuery nodes, as well as equivalent XML values.

a) Two XQuery sequences are equivalent if they have the same number of XQuery items and their corresponding XQuery items are equivalent.

b) Two XQuery atomic values are equivalent if they are identical.

c) Two XQuery nodes ***XN1*** and ***XN2*** are defined to be equivalent if they satisfy all of the following conditions:

i) ***XN1*** and ***XN2*** 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 ***XN1*** is not a document node, then either of the following conditions holds:

1) The **parent** property of *XN1* and the **parent** property of ***XN2*** are both empty.

2) The XQuery node ***P1*** that is the **parent** property of ***XN1*** is equivalent to the XQuery node ***P2*** that is the **parent** property of ***XN2***, and if ***XN1*** and ***XN2*** are not XQuery attribute nodes or XQuery namespace nodes, then ***XN1*** and ***XN2*** have the same ordinal position in the **children** property of ***P1*** and ***P2***, respectively.

iii) Every significant property ***P*** of ***XN1*** is identical to the corresponding property ***P*** of ***XN2***. The significant and insignificant 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-name 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 ***XN1*** is identical to the same property ***P*** of another XQuery node ***XN2*** 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 ***XN1*** and ***XN2*** 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 ***XN1*** equals the number of XQuery items returned by the XQuery accessor of property ***P*** applied to ***XN2***, and there exists a one-to-one mapping of the XQuery items returned by the XQuery accessor of property ***P*** applied to ***XN1*** to the XQuery items returned by the XQuery accessor of property ***P*** applied to ***XN1*** such that corresponding XQuery items are identical.

iii) If ***P*** is the **node-name** property, **type-name** 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 ***XN1*** is identical to the XQuery atomic value that is returned by the XQuery accessor of property ***P*** applied to ***XN2***.

NOTE 82 — 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.

e) An XQuery atomic value and an XQuery node are not equivalent.

Conformance Rules

None.

10.8 Equality operations

This Subclause modifies Subclause 9.11, "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.

10.9 Grouping operations

This Subclause modifies Subclause 9.12, "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.

10.10 Multiset element grouping operations

This Subclause modifies Subclause 9.13, "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.

10.11 Ordering operations

This Subclause modifies Subclause 9.14, "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.

10.12 Determination of namespace URI

Subclause Signature

```
"Determination of namespace URI" [Syntax Rules] (
  Parameter: "QNAME",
  Parameter: "BNFTERM"
) Returns: "NSURI"
```

Function

Determine the XML namespace URI of an XML QName.

Syntax Rules

1) Let ***QN*** be the *QNAME* and let *B* be the *BNFTERM* in an application of the Syntax Rules of this Subclause. The result of the application of this Subclause is *R*, which is returned as *NSURI*.

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, *B* shall be within the scope of one or more <XML namespace declaration>s that contain an <XML namespace prefix> equivalent to ***P*** or ***P*** shall be equivalent to one of **'sqlxml'**, **'xs'**, or **'xsi'**.

            Case:

            A) If *B* is within the scope of one or more <XML namespace declaration>s that contain an <XML namespace prefix> equivalent to ***P***, then:

               I) Let *XND* be the <XML namespace declaration> that contains an <XML namespace prefix> equivalent to ***P*** with innermost scope that includes *B*.

               II) Let *XNDI* be the <XML namespace declaration item> of *XND* that contains an <XML namespace prefix> equivalent to ***P***.

               III) Let *NSURI* be the <XML namespace URI> contained in *XNDI*.

            B) Otherwise,

               Case:

               I) If ***P*** is equivalent to **'sqlxml'**, then let *NSURI* be

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

II) If ***P*** is equivalent to `'xs'`, then let *NSURI* be

`http://www.w3.org/2001/XMLSchema`

III) If ***P*** is equivalent to `'xsi'`, then let *NSURI* be

`http://www.w3.org/2001/XMLSchema-instance`

NOTE 83 — The XML namespace prefixes `fn`, `xs`, and `local`, which are predeclared in [XQuery], if present in an <XML element name> or an <XML attribute name>, need to be in the scope of one or more <XML namespace declaration>s.

b) Otherwise,

Case:

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.

10.13 Construction of an XML element

Subclause Signature

```
"Construction of an XML element" [General Rules] (
  Parameter: "QNAME",
  Parameter: "NAMESPACE",
  Parameter: "ATTRIBUTES",
  Parameter: "CONTENTS",
  Parameter: "OPTION",
  Parameter: "ENCODING"
) Returns: "XMLELEM"
```

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*, let ***NSURI*** be the *NAMESPACE*, let ***ATTRS*** be the *ATTRIBUTES*, let ***CON*** be the *CONTENTS*, let *OPT* be the *OPTION*, and let *ENC* be the *ENCODING* in an application of the General Rules of this Subclause. The result of the application of this Subclause is the result, which is returned as *XMLELEM*.

   NOTE 84 — ***QN*** is an XML 1.1 QName. ***NSURI*** is an XML namespace URI. ***ATTRS*** is a set of XQuery attribute nodes. ***CON*** is a list of values. *ENC* is an indication of whether binary strings are to be encoded in base64 or hex.

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 the null value, 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 the result is an empty XQuery sequence.

   c) Otherwise,

i) For each j, 1 (one) ≤ j ≤ N, such that V_j is not the null value:

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, the General Rules of Subclause 9.8, "Mapping values of SQL data types to values of XML Schema data types", are applied with V_j as *SQLVALUE*, *ENC* as *ENCODING*, "absent" as *NULLS*, and *True* as *CHARMAPPING*; let *XMLV*$_j$ be the *XMLVALUE* returned from the application of those General Rules. ***XMLV***$_j$ is a character string of Unicode characters. Let ***CEC***$_j$ be the result of

```
XMLPARSE (CONTENT XMLV_j PRESERVE WHITESPACE)
```

2) The General Rules of Subclause 10.17, "Removing XQuery document nodes from an XQuery sequence", are applied with ***CEC***$_j$ as *SEQUENCE*; let ***S***$_j$ be the *RESULT* returned from the application of those General Rules.

ii) Let *NILLED* be defined as follows.

Case:

1) If *OPT* is NIL ON NULL and *ALLNULL* is *True*, then *NILLED* is *True*.

2) If *OPT* is NIL ON NO CONTENT and there does not exist at least one j, 1 (one) ≤ j ≤ N, such that V_j 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 ***NILAII*** 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-name** property is **xs:untypedAtomic**.

IV) The **parent** property is empty.

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) The Syntax Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied with the <XML element> or <XML forest> that invoked the current Subclause as *NONTERMI-*

NAL; let ***XSC*** be the *STATICCONTEXT* returned from the application of those Syntax Rules. The validation mode of ***XSC*** is set to an implementation-defined value.

NOTE 85 — ***XSC*** is an XQuery static context.

vi) The General Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied; let ***XDC*** be the *DYNAMICCONTEXT* returned from the application of those General Rules.

NOTE 86 — ***XDC*** is an XQuery dynamic context.

vii) Let ***XSC*** and ***XDC*** be augmented with the following XQuery variables:

1) An XQuery variable **$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 ***QN*** and whose namespace URI is ***NSURI***.

2) An XQuery variable **$SEQ**, whose XQuery formal type notation is **(element of type xs:anyType | attribute of type xs:anySimpleType | text | comment | processing-instruction | xs:anyAtomicType | document { (element of type xs:anyType | comment | processing-instruction | text) * }) *** and whose value is ***S***.

NOTE 87 — 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.

viii) Case:

1) If the SQL-implementation supports Feature X211, "XML 1.1 support", then 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*.

2) Otherwise, let ***EI*** be the result of an XQuery evaluation with XML 1.0 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*.

ix) The result is an XQuery element node that is equivalent to ***EI***.

Conformance Rules

None.

10.14 Concatenation of two XML values

Subclause Signature

```
"Concatenation of two XML values" [General Rules] (
  Parameter: "FIRSTVAL",
  Parameter: "SECONDVAL"
) Returns: "CONCAT"
```

Function

Specify the result of the concatenation of two XML values.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let ***X*** be the *FIRSTVAL* and let ***Y*** be the *SECONDVAL* in an application of the General Rules of this Subclause. The result of the application of this Subclause is ***C***, which is returned as *CONCAT*.

2) Case:

   a) If *X* is the null value, then let *C* be *Y*.

   b) If *Y* is the null value, then let *C* be *X*.

   c) Otherwise, let *C* be the XQuery sequence consisting of every XQuery item of *X*, in order, followed by every XQuery item of *Y*, in order.

3) *C* is the result of the concatenation of *X* and *Y*.

Conformance Rules

None.

10.15 Serialization of an XML value

Subclause Signature

```
"Serialization of an XML value" [General Rules] (
  Parameter: "VALUE",
  Parameter: "SYNTAX",
  Parameter: "TYPE",
  Parameter: "ENCODING",
  Parameter: "BOMDIA",
  Parameter: "VERSION",
  Parameter: "XMLDECLARATION",
  Parameter: "INDENT"
) Returns: "SERIALIZATION"
```

Function

Specify the serialization of an XML value.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let ***V*** be the *VALUE*, let *DC* be the *SYNTAX*, let *DT* be the *TYPE*, let *CS* be the *ENCODING*, let *B* be the *BOMDIA*, let *VP* be the *VERSION*, let *DECL* be the *XMLDECLARATION*, and let *INDENT* be the *INDENT* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *RES*, which is returned as *SERIALIZATION*.

   NOTE 88 — *V* is an XML value. *DC* is either DOCUMENT or CONTENT. *DT* is a string type. *B* is either *True*, *False*, or *Unknown*. *DECL* is either *True*, *False*, or *Unknown*. *INDENT* is either *True* or *False*.

2) If *V* is the null value, then *RES* is the null value and no further General Rules are applied.

3) If *DC* is DOCUMENT and *V* is not an XQuery sequence of length 1 (one) whose sole XQuery item is an XQuery document node whose **children** property contains exactly one XQuery element node, zero or more XQuery comment nodes, and zero or more XQuery processing instruction nodes, then an exception condition is raised: *data exception — not an XML document*.

4) Case:

   a) If *B* is *True*, then let ***BOM*** be **yes**.

   b) If *B* is *False*, then let ***BOM*** be **no**.

c) Otherwise, *B* is *Unknown*, and

Case:

i) If *DT* is a binary string type, then

Case:

1) If *CS* is UTF16, then let ***BOM*** be **yes**.

2) Otherwise, it is implementation-defined whether ***BOM*** is **yes** or **no**.

NOTE 89 — 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.

ii) Otherwise, let ***BOM*** be **no**.

5) Case:

a) If *INDENT* is *True*, then let ***IND*** be "yes".

b) Otherwise, let ***IND*** be "no".

6) Let ***METH*** be an XML 1.0 QName whose XML 1.0 QName prefix is the zero-length string and whose XML 1.0 QName local part is "**xml**".

7) It is implementation-defined whether ***SA*** is **yes**, **no**, or **none**.

8) Case:

a) If *DECL* is *True*, then let ***OXD*** be **no**.

b) If *DECL* is *False*, then let ***OXD*** be **yes**.

c) Otherwise (*DECL* is *Unknown*),

Case:

i) If *DC* is DOCUMENT and any of the following is true:

1) *CS* is neither UTF8 nor UTF16.

2) ***SA*** is not **none**.

3) *VP* is not "1.0".

then let ***OXD*** be **no**.

ii) If *DC* is DOCUMENT and all of the following are true:

1) *CS* is either UTF8 or UTF16.

2) ***SA*** is **none**.

3) *VP* is "1.0".

then it is implementation-defined whether ***OXD*** is **yes** or **no**.

iii) Otherwise, let ***OXD*** be **yes**.

9) Case:

a) If *VP* is "1.0", then let ***UN*** be **no**.

b) Otherwise, it is implementation-defined whether ***UN*** is **yes** or **no**.

10) Let ***CSE***, ***DP***, ***DS***, ***EUA***, ***ICT***, ***MT***, ***NF***, and ***UCM*** be implementation-dependent values that are permitted values for the cdata-section-elements, doctype-public, doctype-system, escape-uri-attributes, include-content-type, media-type, normalization-form, and use-character-maps serialization parameters, respectively, as defined in Section 3, "Serialization Parameters", of [Serialization].

11) Case:

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

i) Let ***CV*** be the result of applying Section 5, "XML output method", of [Serialization] to *V* with the following values for the serialization parameters:

1) ***BOM*** as the byte-order-mark.

2) ***CSE*** as the cdata-section-elements.

3) ***DP*** as the doctype-public.

4) ***DS*** as the doctype-system.

5) ***CS*** as the encoding.

6) ***EUA*** as the escape-uri-attributes.

7) ***ICT*** as the include-content-type.

8) ***IND*** as the indent.

9) ***MT*** as the media-type.

10) ***METH*** as the method.

11) ***NF*** as the normalization-form.

12) ***OXD*** as the omit-xml-declaration.

13) ***SA*** as the standalone.

14) ***UN*** as the undeclare-prefixes.

15) ***UCM*** as the use-character-maps.

16) ***VP*** as the version.

If an XQuery serialization error occurs during the application of Section 5, "XML output method", of [Serialization] to *V*, then an exception condition is raised: *data exception — XQuery serialization error*.

ii) The General Rules of Subclause 10.2, "Store assignment", are applied with a temporary site *TS* whose declared type is *DT* as *TARGET*, *CV* as *VALUE*, and the null value as *PASSING*.

iii) Let *RES* be the value of *TS*.

b) Otherwise (*DT* is a binary string type):

i) Let *CV* be a binary string equivalent to the result of applying Section 5, "XML output method", of [Serialization] to *V* with the following values for the serialization parameters:

1) ***BOM*** as the byte-order-mark.
2) ***CSE*** as the cdata-section-elements.
3) ***DP*** as the doctype-public.
4) ***DS*** as the doctype-system.
5) ***CS*** as the encoding.
6) ***EUA*** as the escape-uri-attributes.
7) ***ICT*** as the include-content-type.
8) ***IND*** as the indent.
9) ***MT*** as the media-type.
10) ***METH*** as the method.
11) ***NF*** as the normalization-form.
12) ***OXD*** as the omit-xml-declaration.
13) ***SA*** as the standalone.
14) ***UN*** as the undeclare-prefixes.
15) ***UCM*** as the use-character-maps.
16) ***VP*** as the version.

If an XQuery serialization error occurs during the application of Section 5, "XML output method", of [Serialization] to *V*, then an exception condition is raised: *data exception — XQuery serialization error*.

ii) The General Rules of Subclause 10.2, "Store assignment", are applied with a temporary site *TS* whose declared type is *DT* as *TARGET*, *CV* as *VALUE*, and the null value as *PASSING*.

iii) Let *RES* be the value of *TS*.

Conformance Rules

None.

10.16 Parsing a string as an XML value

Subclause Signature

```
"Parsing a string as an XML value" [General Rules] (
  Parameter: "SYNTAX",
  Parameter: "TEXT",
  Parameter: "OPTION"
) Returns: "XMLPARSE"
```

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*, let *V* be the *TEXT*, and let *WO* be the *OPTION* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *D*, which is returned as *XMLPARSE*.

   NOTE 90 — *DC* is either DOCUMENT or CONTENT. *WO* is either PRESERVE WHITESPACE or STRIP WHITESPACE.

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 any of the following is true:

   a) The character string is a textual XML 1.0 document.

   b) The character string conforms to the definition of a well-formed external parsed entity as defined in [XML 1.0], as modified by [Namespaces 1.0].

5) A string is called a *textual XML 1.1 content* if any of the following is true:

   a) The character string is a textual XML 1.1 document.

   b) The character string conforms to the definition of a well-formed external parsed entity as defined in [XML 1.1], as modified by [Namespaces 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 91 — 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 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**.

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 92 — 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 [XQueryDM] section 6, "Nodes".

10) ***D*** is the result of parsing *V* as an XML value.

Conformance Rules

None.

10.17 Removing XQuery document nodes from an XQuery sequence

Subclause Signature

```
"Removing XQuery document nodes from an XQuery sequence" [General Rules] (
  Parameter: "SEQUENCE"
) Returns: "RESULT"
```

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 the General Rules of this Subclause. The result of the application of this Subclause is ***R***, which is returned as *RESULT*.

2) Case:

   a) If *S* is the null value, then let ***R*** be the null value.

   b) Otherwise:

      i) Let *N* be the number of XQuery items in *S*.

      ii) Let $\boldsymbol{I}_j$, 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 $\boldsymbol{I}_j$ is an XQuery document node, then let $\boldsymbol{C}_j$ be the XQuery sequence that is the **children** property of $\boldsymbol{I}_j$.

         2) Otherwise, let $\boldsymbol{C}_j$ be $\boldsymbol{I}_j$.

      iv) Let ***R*** be the concatenation of the XQuery sequences $\boldsymbol{C}_j$ in order from *j* = 1 (one) through *j* = *N*.

   c) ***R*** is the result of this Subclause.

Conformance Rules

None.

10.18 Constructing a copy of an XML value

Subclause Signature

```
"Constructing a copy of an XML value" [General Rules] (
  Parameter: "VALUE"
) Returns: "COPY"
```

Function

Construct a copy of an XML value.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let ***V*** be the *VALUE* in an application of the General Rules of this Subclause. The result of the application of this Subclause is ***R***, which is returned as *COPY*.

   NOTE 93 — ***V*** is a value of an XML type.

2) Let *N* be the number of XQuery items in *V*.

3) Let $\boldsymbol{I}_j$, 1 (one) ≤ *j* ≤ *N*, be an enumeration in order of the XQuery items in *V*.

4) For each *j*, 1 (one) ≤ *j* ≤ *N*, let $\boldsymbol{C}_j$ be defined as follows.

   Case:

   a) If $\boldsymbol{I}_j$ is an XQuery atomic value, then let $\boldsymbol{C}_j$ be $\boldsymbol{I}_j$.

   b) Otherwise, let $\boldsymbol{C}_j$ be a value of XML type that is equivalent to $\boldsymbol{I}_j$ except that the **parent** property (if any) of $\boldsymbol{C}_j$ is **empty** and the XQuery node identity of $\boldsymbol{C}_j$ is different from the XQuery node identity of $\boldsymbol{I}_j$.

5) Let ***R*** be the XQuery sequence enumerated by $\boldsymbol{C}_j$, 1 (one) ≤ *j* ≤ *N*.

Conformance Rules

None.

10.19 Constructing an unvalidated XQuery document node

Subclause Signature

```
"Constructing an unvalidated XQuery document node" [General Rules] (
  Parameter: "XQUERYDOCNODE"
) Returns: "UNVALIDATEDDOCNODE"
```

Function

Convert an XQuery document node to a value of type XML(CONTENT(UNTYPED)).

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let ***XDN*** be the *XQUERYDOCNODE* in an application of the General Rules of this Subclause. The result of the application of this Subclause is ***R***, which is returned as *UNVALIDATEDDOCNODE*.

2) Let ***R*** be an XQuery document node that is equivalent to ***XDN*** except:

   a) The XQuery node identity of ***R*** is different from the XQuery node identity of ***XDN***.

   b) The **type-name** property of every XQuery element node in the XQuery tree rooted by ***R*** is `xs:untyped`.

   c) The **nilled** property of every XQuery element node in the XQuery tree rooted by ***R*** is `false`.

   d) The **type-name** property of every XQuery attribute node in the XQuery tree rooted by ***R*** is `xs:untypedAtomic`.

3) ***R*** is the result of this Subclause.

Conformance Rules

None.

10.20 Creation of an XQuery expression context

Subclause Signature

```
"Creation of an XQuery expression context" [Syntax Rules] (
  Parameter: "NONTERMINAL"
) Returns: "STATICCONTEXT"

"Creation of an XQuery expression context" [General Rules] () Returns: "DYNAMICCONTEXT"
```

Function

Create an XQuery expression context.

Syntax Rules

1) Let *BNF* be the *NONTERMINAL* in an application of the Syntax Rules of this Subclause. The result of the application of this Subclause is ***XSC***, which is returned as *STATICCONTEXT*.

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 initial value".

   b) The **statically known namespaces** component of ***XSC*** is augmented with the XML namespace prefix/URI pair consisting of

      ```
      ('sqlxml', "http://standards.iso.org/iso/9075/2003/sqlxml")
      ```

   c) ***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?".

   d) If *BNF* is specified, then:

      i) The **statically known 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 **statically known namespaces** component of ***XSC***, either,

            Case:

            A) If *XNP* is previously defined, then overwriting the previously defined namespace.

            B) Otherwise, augmenting the **statically known namespaces** component of ***XSC***.

NOTE 94 — The scope of <XML namespace declaration>s is defined in the various Subclauses that reference that nonterminal symbol, such as Subclause 6.14, "<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 form "NO DEFAULT", then the **default element/type namespace** component of ***XSC*** is set to empty.

e) The in-scope schema definitions component of ***XSC*** (i.e., the in-scope schema types, 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) The General Rules of this Subclause are applied without any symbolic arguments.The result of the application of this Subclause is ***XDC***, which is returned as *DYNAMICCONTEXT*.

2) 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 initial value" and "Can be overwritten or augmented by implementation?".

3) ***XDC*** is the result of this Subclause.

Conformance Rules

None.

10.21 Determination of an XQuery formal type notation

Subclause Signature

```
"Determination of an XQuery formal type notation" [Syntax Rules] (
  Parameter: "SOURCE",
  Parameter: "CONTEXT"
) Returns: "FORMALTYPE"
```

Function

Determine the XQuery formal type notation of an XQuery variable in the XQuery static context.

Syntax Rules

1) Let *S* be the *SOURCE* and let *C* be the *CONTEXT* in an application of the Syntax Rules of this Subclause. The result of the application of this Subclause is ***XFTN*** or an implementation-defined XML Schema subtype of ***XFTN*** that describes *S*, which is returned as *FORMALTYPE*.

2) Let *SD* be the declared type of *S*.

3) Case:

   a) If *C* is *True*, then let ***SUFFIX*** be the zero-length string.

   b) If *SD* is XML(SEQUENCE), then let ***SUFFIX*** be "*******"

   c) If *S* is a column reference that references a known not null column or a <value expression> for which the SQL-implementation can deduce via an implementation-defined rule that the value of *S* cannot be null, then let ***SUFFIX*** be the zero-length string.

   d) Otherwise, let ***SUFFIX*** be "**?**".

4) If *SD* is XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)), then:

   a) Let ***XS*** be the XML Schema identified by the registered XML Schema descriptor included in the XML type descriptor of *SD*.

   b) Case:

      i) If the XML type descriptor of *SD* includes the XML namespace URI and the XML NCName of a global element declaration schema component ***GE***, then let ***ET*** be the XML QName of ***GE***.

      ii) If the XML type descriptor of *SD* includes the XML namespace URI ***NS***, then let *N* be the number of global element declaration schema components included in ***NS*** and let $\mathbf{ET}_i$, 1 (one) ≤ *i* ≤ *N*, be the XML QNames of those global element declaration schema components.

      iii) Otherwise, let *N* be the number of global element declaration schema components included in ***XS*** and let $\mathbf{ET}_i$, 1 (one) ≤ *i* ≤ *N*, be the XML QNames of those global element declaration schema components.

5) Let ***XFTN*** be the XQuery formal type notation determined as follows.

Case:

a) If *SD* is XML(DOCUMENT(UNTYPED)), then let ***XFTN*** be

```
document { (comment | processing-instruction)*,
                element * of type xs:untyped,
                (comment | processing-instruction)* } SUFFIX
```

b) If *SD* is XML(DOCUMENT(ANY)),then let ***XFTN*** be

```
document { (comment | processing-instruction)*,
                element * of type xs:anyType,
                (comment | processing-instruction)* } SUFFIX
```

c) If *SD* is XML(DOCUMENT(XMLSCHEMA)), then:

i) If the type descriptor of *SD* includes the XML namespace URI and the XML NCName of a global element declaration schema component, then let ***XFTN*** be

```
document { (comment | processing-instruction)*,
                element * of type ET,
                (comment | processing-instruction)* } SUFFIX
```

ii) Otherwise, let ***XFTN*** be

```
document { (comment | processing-instruction)*,
                (element * of type ET1 | element * of type ET2 | ... | element
 * of type ETN,
                (comment | processing-instruction)* } SUFFIX
```

d) If *SD* is XML(CONTENT(UNTYPED)), then let ***XFTN*** be

```
document { ( element * of type xs:untyped | comment
                | processing-instruction | text)* } SUFFIX
```

e) If *SD* is XML(CONTENT(ANY)), then let ***XFTN*** be

```
document { ( element * of type xs:anyType | comment
                | processing-instruction | text)* } SUFFIX
```

f) If *SD* is XML(CONTENT(XMLSCHEMA)), then:

i) If the type descriptor of *SD* includes the XML namespace URI and the XML NCName of a global element declaration schema component, then let ***XFTN*** be

```
document { ( element * of type ET |
                comment | processing-instruction)* } SUFFIX
```

ii) Otherwise, let ***XFTN*** be

```
document { ( element * of type ET1 | element * of type ET2 | ... | element *
of type ETN |
                comment | processing-instruction)* } SUFFIX
```

g) If *SD* is XML(SEQUENCE), then let ***XFTN*** be

```
( element * of type xs:anyType | attribute of type xs:anySimpleType |
  text | comment | processing-instruction | xs:anyAtomicType |
  document { ( element * of type xs:anyType | comment |
  processing-instruction | text )* } ) SUFFIX
```

h) If *SD* is an SQL predefined type, then let *ENC* be an indication that the binary strings are to be encoded in hex; the General Rules of Subclause 9.5, "Mapping SQL data types to XML Schema data types", are applied with *SD* as *SQLTYPE*, *ENC* as *ENCODING*, and "absent" as *NULLS*; let ***XMLT*** be the *SCHEMA TYPE* returned from the application of those General Rules. 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 "**xs:yearMonthDuration *SUFFIX***".

ii) If *SD* is a day-time interval, then let ***XFTN*** be "**xs:dayTimeDuration *SUFFIX***".

iii) If *SD* is an exact numeric type with scale 0 (zero), then let ***XFTN*** be "**xs:integer *SUFFIX***".

iv) Otherwise, let ***XFTN*** be "***PT SUFFIX***".

6) ***XFTN***, or an implementation-defined XML Schema subtype of ***XFTN*** that describes *S*, is the result of this Subclause.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

10.22 Validating an XQuery document or element node

Subclause Signature

```
"Validating an XQuery document or element node" [General Rules] (
  Parameter: "ITEM",
  Parameter: "SCHEMA"
) Returns: "RESULT"
```

Function

Validate an XQuery document or an XQuery element node.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let ***V*** be the *ITEM* and let *SXS* be the *SCHEMA* in an application of the General Rules of this Subclause. The result of the application of this Subclause is ***EI***, which is returned as *RESULT*.

2) The Syntax Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied with the null value as *NONTERMINAL*; let ***XSC*** be the *STATICCONTEXT* returned from the application of those Syntax Rules.

   NOTE 95 — ***XSC*** is an XQuery static context.

3) The General Rules of Subclause 10.20, "Creation of an XQuery expression context", are applied; let ***XDC*** be the *DYNAMICCONTEXT* returned from the application of those General Rules.

   NOTE 96 — ***XDC*** is an XQuery dynamic context.

4) *SXS* is included in the in-scope schema definitions component of ***XSC***.

5) ***XSC*** and ***XDC*** are augmented with an XQuery variable **$SEQ**, whose XQuery formal type notation is

```
(element of type xs:anyType | attribute of type xs:anySimpleType |
 text | comment | processing-instruction | xs:anyAtomicType |
 document-node { ( element of type xs:anyType | comment |
                   processing-instruction | text )* } )*
```

   and whose value is ***V***.

6) Case:

a) If the SQL-implementation supports Feature X211, "XML 1.1 support", then 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

```
validate strict { $SEQ }
```

If an XQuery error occurs during the evaluation, then let *STAT* be FAILURE; otherwise, let *STAT* be SUCCESS.

b) Otherwise, let ***EI*** be the result of an XQuery evaluation with XML 1.0 lexical rules, using ***XSC*** and ***XDC*** as the XQuery expression context, of the XQuery expression

```
validate strict { $SEQ }
```

If an XQuery error occurs during the evaluation, then let *STAT* be FAILURE; otherwise, let *STAT* be SUCCESS.

7) If *STAT* is FAILURE, then let ***EI*** be the null value.

Conformance Rules

None.

11 Additional common elements

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

11.1 <routine invocation>

This Subclause modifies Subclause 10.4, "<routine invocation>", in ISO/IEC 9075-2.

Function

Invoke an SQL-invoked routine.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 3)b)i) If P_i is an input SQL parameter or both an input SQL parameter and an output SQL parameter, then

   Case:

   a) If T_i is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with V_i as *VALUE*, a temporary site *ST* whose declared type is T_i as *TARGET*, and the <XML passing mechanism> of P_i as *PASSING*.

   b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with V_i as *VALUE*, a temporary site *ST* whose declared type is T_i as *TARGET*, and the null value as *PASSING*.

   Let CPV_i be the value of *ST*.

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) It is implementation-defined whether XDO_i is INCLUDING XMLDECLARATION or EXCLUDING XMLDECLARATION.

d) 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.

e) CPV_i is replaced by the result of

```
BOMi || XMLSERIALIZE (XOi CPVi AS XDTi XDOi)
```

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)i)i)4)B)III)2)b) If *CRT* is an XML type, then:

a) Let *XO* be the associated XML option of the result of *R*.

b) It is implementation-defined whether *XWO* is STRIP WHITESPACE or PRESERVE WHITESPACE.

c) Let *RDI* be the result of

```
XMLPARSE (XO ERDI XWO)
```

5) Insert before GR 8)j)iii)3)C) If T_i is an XML type, then:

a) Let XO_i be the associated XML option of P_i.

b) It is implementation-defined whether XWO_i is STRIP WHITESPACE or PRESERVE WHITESPACE.

c) CPV_i is set to the result of

```
XMLPARSE (XOi EVi XWOi)
```

6) Replace GR 9)a)iii) Let the result of the <routine invocation> be

Case:

a) If *ERDT* is an XML type, then the value of a target *T* of declared type *ERDT* after the General Rules of Subclause 10.2, "Store assignment", are applied with *RV* as *VALUE*, a temporary site *ST* whose declared type is *ERDT* as *TARGET*, and the <XML passing mechanism> of the <returns clause> of *R* as *PASSING*.

b) Otherwise, the value of a target *T* of declared type *ERDT* after the General Rules of Subclause 10.2, "Store assignment", are applied with *RV* as *VALUE*, a temporary site *ST* whose declared type is *ERDT* as *TARGET*, and the null value as *PASSING*.

7) Replace GR 9)b)ii)1)B)V)1)b) Case:

   a) If *EDT* is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with the *I*-th element of *A* as *TARGET*, the value of CPV_i as *VALUE*, and the <XML passing mechanism> of P_i as *PASSING*.

   b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with the *I*-th element of *A* as *TARGET*, the value of CPV_i as *VALUE*, and the null value as *PASSING*.

8) Replace GR 9)b)ii)1)B)V)2)c) Case:

   a) If *EDT* is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with the *I*-th element of *A* as *TARGET*, the value of CPV_i as *VALUE*, and the <XML passing mechanism> of P_i as *PASSING*.

   b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with the *I*-th element of *A* as *TARGET*, the value of CPV_i as *VALUE*, and the null value as *PASSING*.

9) Replace GR 9)b)ii)2) Otherwise,

   Case:

   a) If the declared type of P_i is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with TS_i as *TARGET*, the value of CPV_i as *VALUE*, and the <XML passing mechanism> of P_i as *PASSING*.

   b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with TS_i as *TARGET*, the value of CPV_i as *VALUE*, and the null value as *PASSING*.

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 aggregate> is specified, then:

   a) If <XML returning clause> is not specified, then it is implementation-defined whether RETURNING CONTENT or RETURNING SEQUENCE is implicit.

   b) Case:

      i) If RETURNING CONTENT is specified or implied, then

         Case:

         1) If the declared type of the <XML value expression> is XML(DOCUMENT(UNTYPED)) or XML(CONTENT(UNTYPED)), then the declared type of <XML aggregate> is XML(CONTENT(UNTYPED)).

         2) If the declared type of the <XML value expression> is XML(DOCUMENT(XMLSCHEMA)) or XML(CONTENT(XMLSCHEMA)), then let *XVT* be the declared type of <XML value expression>, let *XS* be the registered XML Schema descriptor included in the XML type descriptor of *XVT*, let *NS* be the XML Namespace URI included in the XML type descriptor of *XVT*, if any, and let *EN* be the XML NCName of a global element declaration schema component included in the XML type descriptor of *XVT*, if any. The declared type of <XML aggregate> is XML(CONTENT(XMLSCHEMA)) whose XML type descriptor includes *XS*, if *NS* is defined, then *NS*, and, if *EN* is defined, then *EN*.

         3) Otherwise, the declared type of <XML aggregate> is XML(CONTENT(ANY)).

      ii) Otherwise, the declared type of <XML aggregate> is XML(SEQUENCE).

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

d) If <sort specification list> is specified, then let *SSL* be "ORDER BY <sort specification list>"; otherwise, let *SSL* be the zero-length string.

e) If RETURNING CONTENT is specified or implied, then

i) Let *RT* be the declared type of <XML aggregate>.

ii) <XML aggregate> is equivalent to

```
CAST (
  XMLDOCUMENT (
    XMLAGG (
      XVE SSL
      RETURNING SEQUENCE )
    RETURNING CONTENT )
  AS RT )
```

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 3)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) For all *i*, 2 ≤ *i* ≤ *N*, the General Rules of Subclause 10.14, "Concatenation of two XML values", are applied with V_{i-1} as *FIRSTVAL* and the value of the first column of R_i as *SECONDVAL*; let V_i be the *CONCAT* returned from the application of those General Rules.

3) The result is V_N.

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, "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, "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 or casting an XML value.

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.

11.6 <XML valid according to clause>

Function

Indicate a registered XML Schema, and (optionally) an XML namespace of that registered XML Schema, and (optionally) a global element declaration schema component of that registered XML Schema.

Format

```
<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 URI> [ <XML valid schema location> ]
  | NO NAMESPACE [ <XML valid schema location> ]

<XML valid target namespace URI> ::=
  <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> ::=
    <XML valid element name specification>
  | <XML valid element namespace specification>
      [ <XML valid element name specification> ]

<XML valid element name specification> ::=
  ELEMENT <XML valid element name>

<XML valid element namespace specification> ::=
    NO NAMESPACE
  | NAMESPACE <XML valid element namespace URI>

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

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

Syntax Rules

1) If <XML valid according to identifier> is specified, then the <registered XML Schema name> shall identify a registered XML Schema *RXS*. Let *NSURI* be the target namespace URI of *RXS*.

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

   a) Case:

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

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

   b) Case:

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

         Case:

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

            Case:

            A) If there exists more than one such registered XML Schema, then let *XSDN* be a <registered XML Schema name> chosen 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, the same <registered XML Schema name> will be chosen.

            B) Otherwise, let *XSD* be the registered XML Schema descriptor that includes *NSURI* as its target namespace URI and *XVSL* as its schema location URI. Let *XSDN* be the <registered XML Schema name> included in *XSD*.

         2) Otherwise, a <registered XML Schema name> *XSDN* is chosen 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, the same <registered XML Schema name> will be chosen.

            NOTE 97 — This does not say that the algorithm must choose a <registered XML Schema name> that identifies a registered XML Schema. One possibility for the implementation-defined algorithm is to choose a <registered XML Schema name> that does not identify a registered XML Schema, resulting in a syntax error.

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

         Case:

         1) If there exists more than one such registered XML Schema, then let *XSDN* be a <registered XML Schema name> chosen 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, the same <registered XML Schema name> will be chosen.

2) Otherwise, let *XSD* be the registered XML Schema descriptor that includes *NSURI* as its target namespace URI. Let *XSDN* be the <registered XML Schema name> included in *XSD*.

c) *XSDN* shall identify a registered XML Schema *RXS*.

3) *RXS* is the *indicated registered XML schema*.

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

a) Case:

i) If <XML valid element namespace specification> is specified, then

Case:

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

2) Otherwise, let *ENSURI* be <XML valid element namespace URI>.

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

b) It is implementation-defined whether *RXS* shall have a namespace, among its unordered collection of namespaces, whose namespace URI is identical to *ENSURI*, as defined by [Namespaces].

c) *ENSURI* is the *indicated XML namespace*.

d) If <XML valid element name specification> is specified, then:

i) Let *EN* be the <XML valid element name>.

ii) Let *GEDSC* be the global element declaration schema component whose namespace URI is identical to *ENSURI*, as defined by [Namespaces], and whose XML NCName is *EN*. *GEDSC* is the *indicated global element declaration schema component*.

iii) It is implementation-defined whether *RXS* shall have a global element declaration schema component whose namespace URI is identical to *ENSURI*, as defined by [Namespaces], and whose XML NCName is *EN*.

Access Rules

1) Case:

a) If the <XML valid according to clause> 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 *RXS*.

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

General Rules

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

a) If *RXS* does not have a namespace, among its unordered collection of namespaces, whose namespace URI is identical to *ENSURI*, as defined by [Namespaces], then an exception condition is raised: *data exception — element namespace not declared*.

b) If <XML valid element name specification> is specified and *RXS* does not have a global element declaration schema component whose namespace URI is identical to *ENSURI*, as defined by [Namespaces], and whose XML NCName is *EN*, then an exception condition is raised: *data exception — global element not declared*.

Conformance Rules

None.

12 Schema definition and manipulation

This Clause modifies Clause 11, "Schema definition and manipulation", in ISO/IEC 9075-2.

12.1 <column definition>

This Subclause modifies Subclause 11.4, "<column definition>", in ISO/IEC 9075-2.

Function

Define a column of a base table.

Format

```
<generation expression> ::=
  <left paren> [ WITH <XML 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(DOCUMENT(UNTYPED)) type”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(DOCUMENT(UNTYPED)) type or a distinct type whose source type is the XML(DOCUMENT(UNTYPED)) type.

5) Insert this CR Without Feature X252, “Persistent XML values of XML(DOCUMENT(ANY)) type”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(DOCUMENT(ANY)) type or a distinct type whose source type is the XML(DOCUMENT(ANY)) type.

6) Insert this CR Without Feature X256, “Persistent XML values of XML(DOCUMENT(XMLSCHEMA)) type”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(DOCUMENT(XMLSCHEMA)) type or a distinct type whose source type is the XML(DOCUMENT(XMLSCHEMA)) type.

7) Insert this CR Without Feature X253, “Persistent XML values of XML(CONTENT(UNTYPED)) type”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(CONTENT(UNTYPED)) type or a distinct type whose source type is the XML(CONTENT(UNTYPED)) type.

8) Insert this CR Without Feature X254, “Persistent XML values of XML(CONTENT(ANY)) type”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(CONTENT(ANY)) type or a distinct type whose source type is the XML(CONTENT(ANY)) type.

9) Insert this CR Without Feature X257, “Persistent XML values of XML(CONTENT(XMLSCHEMA)) type”, conforming SQL language shall not contain a <column definition> whose declared type is based on either the XML(CONTENT(XMLSCHEMA)) type or a distinct type whose source type is the XML(CONTENT(XMLSCHEMA)) type.

10) 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 options> 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 <alter column data type clause>

This Subclause modifies Subclause 11.19, "<alter column data type clause>", in ISO/IEC 9075-2.

Function

Change the declared type of a column.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert after SR 15)jj) If *DTC* is an XML type, then:

   a) Let *PDTC* be the primary XML type modifier of *DTC*, let *SDTC* be the secondary XML type modifier of *DTC* (if any), let *SCHDTC* be the registered XML Schema of *DTC* (if any), let *NSDTC* be the XML namespace URI of *DTC* (if any), and let *EDTC* be the XML NCName of the global element of *DTC* (if any).

   b) Let *PD* be the primary XML type modifier of *D*, let *SD* be the secondary XML type modifier of *D* (if any), let *SCHD* be the registered XML Schema of *D* (if any), let *NSD* be the XML namespace URI of *D* (if any), and let *ED* be the XML NCName of the global element of *D* (if any).

   c) If *PDTC* is CONTENT, then *PD* shall be CONTENT or SEQUENCE.

   d) If *SDTC* is UNTYPED, then *SD* shall be UNTYPED or ANY.

   e) If *SDTC* is XMLSCHEMA, then *SD* shall be XMLSCHEMA or ANY.

   f) If *SCHDTC* is defined, then *SCHD* shall be either identical to *SCHDTC* or not defined.

   g) If *NSDTC* is defined, then *NSD* shall be either identical to *NSDTC* as defined by [Namespaces] or not defined.

   h) If *EDTC* is defined, then *ED* shall be either equivalent to *EDTC* or not defined.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert after CR 1) Without Feature X410, "Alter column data type: XML type", conforming SQL language shall not specify an <alter column data type clause> that contains a <data type> that specifies an XML type.

12.4 <view definition>

This Subclause modifies Subclause 11.32, "<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(DOCUMENT(UNTYPED)) type", conforming SQL language shall not contain a <view definition> that defines a column whose declared type is based on either the XML(DOCUMENT(UNTYPED)) type or a distinct type whose source type is the XML(DOCUMENT(UNTYPED)) type.

3) Insert this CR Without Feature X252, "Persistent XML values of XML(DOCUMENT(ANY)) type", conforming SQL language shall not contain a <view definition> that defines a column whose declared type is based on either the XML(DOCUMENT(ANY)) type or a distinct type whose source type is the XML(DOCUMENT(ANY)) type.

4) Insert this CR Without Feature X256, "Persistent XML values of XML(DOCUMENT(XMLSCHEMA)) type", conforming SQL language shall not contain a <view definition> that defines a column whose declared type is based on either the XML(DOCUMENT(XMLSCHEMA)) type or a distinct type whose source type is the XML(DOCUMENT(XMLSCHEMA)) type.

5) Insert this CR Without Feature X253, "Persistent XML values of XML(CONTENT(UNTYPED)) type", conforming SQL language shall not contain a <view definition> that defines a column whose declared

type is based on either the XML(CONTENT(UNTYPED)) type or a distinct type whose source type is the XML(CONTENT(UNTYPED)) type.

6) Insert this CR Without Feature X254, "Persistent XML values of XML(CONTENT(ANY)) type", conforming SQL language shall not contain a <view definition> that defines a column whose declared type is based on either the XML(CONTENT(ANY)) type or a distinct type whose source type is the XML(CONTENT(ANY)) type.

7) Insert this CR Without Feature X257, "Persistent XML values of XML(CONTENT(XMLSCHEMA)) type", conforming SQL language shall not contain a <view definition> that defines a column whose declared type is based on either the XML(CONTENT(XMLSCHEMA)) type or a distinct type whose source type is the XML(CONTENT(XMLSCHEMA)) type.

8) Insert this CR Without Feature X255, "Persistent XML values of XML(SEQUENCE) type", conforming SQL language shall not contain a <view definition> that defines a column whose declared type is based on either the XML(SEQUENCE) type or a distinct type whose source type is the XML(SEQUENCE) type.

12.5 <assertion definition>

This Subclause modifies Subclause 11.47, "<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>
      [ <constraint characteristics> ]
```

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>.

12.6 <user-defined type definition>

This Subclause modifies Subclause 11.51, "<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 of XML type", conforming SQL language shall not contain a <representation> that is a <predefined type> that is an XML type.

12.7 <attribute definition>

This Subclause modifies Subclause 11.52, "<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.

12.8 <SQL-invoked routine>

This Subclause modifies Subclause 11.60, "<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 ]
      [ DEFAULT <parameter default> ]
      [ <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 9)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 9)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 24)d)iii)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 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 24)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 24)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 24)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 24)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 it is implementation-defined whether BY REF or 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 it is implementation-defined whether BY REF or 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)r) 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)r) 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)r) For every SQL parameter whose <SQL parameter declaration> contains an explicit or implicit <XML passing mechanism>, the routine descriptor includes an indication of that <XML passing mechanism>.

4) Insert after GR 3)s) If *R* is an SQL function, then the routine descriptor includes 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.

12.9 <user-defined cast definition>

This Subclause modifies Subclause 11.63, "<user-defined cast definition>", in ISO/IEC 9075-2.

Function

Define a user-defined cast.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert this SR Neither *SDT* nor *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 <externally-invoked procedure>

This Subclause modifies Subclause 13.3, "<externally-invoked procedure>", in ISO/IEC 9075-2.

Function

Define an externally-invoked procedure.

Format

No additional Format items.

Syntax Rules

1) Insert into SR 10)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_SERIALIZATION_ERROR:
  constant SQLSTATE_TYPE := "2200W";
DATA_EXCEPTION_XQUERY_SEQUENCE_CANNOT_BE_VALIDATED:
  constant SQLSTATE_TYPE := "2201J";
DATA_EXCEPTION_XQUERY_DOCUMENT_NODE_CANNOT_BE_VALIDATED:
```

```
    constant SQLSTATE_TYPE := "2201K";
DATA_EXCEPTION_NO_XML_SCHEMA_FOUND:
    constant SQLSTATE_TYPE := "2201L";
DATA_EXCEPTION_ELEMENT_NAMESPACE_NOT_DECLARED:
    constant SQLSTATE_TYPE := "2201M";
DATA_EXCEPTION_GLOBAL_ELEMENT_NOT_DECLARED:
    constant SQLSTATE_TYPE := "2201N";
DATA_EXCEPTION_NO_XML_ELEMENT_WITH_SPECIFIED_QNAME:
    constant SQLSTATE_TYPE := "2201P";
DATA_EXCEPTION_NO_XML_ELEMENT_WITH_SPECIFIED_NAMESPACE:
    constant SQLSTATE_TYPE := "2201Q";
DATA_EXCEPTION_VALIDATION_FAILURE:
    constant SQLSTATE_TYPE := "2201R";
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";
```

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 13.4, "<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 13.5, "Data type correspondences", in ISO/IEC 9075-2.

Function

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

Tables

Table 5, "Data type correspondences for Ada", modifies Table 19, "Data type correspondences for Ada", in [ISO9075-2].

Table 5 — Data type correspondences for Ada

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

Table 6, "Data type correspondences for C", modifies Table 20, "Data type correspondences for C", in [ISO9075-2].

Table 6 — Data type correspondences for C

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

Table 7, "Data type correspondences for COBOL", modifies Table 21, "Data type correspondences for COBOL", in [ISO9075-2].

Table 7 — Data type correspondences for COBOL

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

Table 8, "Data type correspondences for Fortran", modifies Table 22, "Data type correspondences for Fortran", in [ISO9075-2].

Table 8 — Data type correspondences for Fortran

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

Table 9, "Data type correspondences for M", modifies Table 23, "Data type correspondences for M", in [ISO9075-2].

Table 9 — Data type correspondences for M

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

Table 10, "Data type correspondences for Pascal", modifies Table 24, "Data type correspondences for Pascal", in [ISO9075-2].

Table 10 — Data type correspondences for Pascal

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

Table 11, "Data type correspondences for PL/I", modifies Table 25, "Data type correspondences for PL/I", in [ISO9075-2].

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.

14 Data manipulation

This Clause modifies Clause 14, "Data manipulation", in ISO/IEC 9075-2.

14.1 <fetch statement>

This Subclause modifies Subclause 14.5, "<fetch statement>", in ISO/IEC 9075-2.

Function

Position a standing cursor on a specified row of the standing cursor's result set and retrieve values from that row.

Format

```
<fetch target list> ::=
  <target specification 1> [ { <comma> <target specification 1> }... ]

<target specification 1> ::=
  <target specification> [ <XML passing mechanism> ]
```

Syntax Rules

1) Insert this SR If a <target specification 1> *TS1* contains an <XML passing mechanism>, then:

   a) The declared type of the <target specification> *TS2* contained in *TS1* shall be an XML type.

   b) *TS2* shall not be a <host parameter specification>, an <embedded variable specification>, or a <dynamic parameter specification>.

2) Insert this SR If the declared type of the <target specification> *TS2* contained in a <target specification 1> *TS1* is an XML type, *TS2* is not a <host parameter specification>, an <embedded variable specification>, or a <dynamic parameter specification>, and *TS1* does not contain an <XML passing mechanism>, then it is implementation-defined whether an <XML passing mechanism> of BY REF or BY VALUE is implicit.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 5)a)i) If *TS* is an <SQL parameter reference>, then

Case:

a) If the declared type of *TS* is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with *TS* as *TARGET*, the current row as *VALUE*, and the <XML passing mechanism> of *TS* as *PASSING*.

b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with *TS* as *TARGET*, the current row as *VALUE*, and the null value as *PASSING*.

2) Replace GR 5)b)i)1)B)V)1)b) The *I*-th element of *A* is set to

Case:

a) If *EDT* is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with *I*-th element of *A* as *TARGET*, SV_i as *VALUE*, and the <XML passing mechanism> of TV_i as *PASSING*.

b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with *I*-th element of *A* as *TARGET*, SV_i as *VALUE*, and the null value as *PASSING*.

3) Replace GR 5)b)i)1)B)V)2)c) The *I*-th element of *A* is set to

Case:

a) If *EDT* is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with *I*-th element of *A* as *TARGET*, SV_i as *VALUE*, and the <XML passing mechanism> of TV_i as *PASSING*.

b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with *I*-th element of *A* as *TARGET*, SV_i as *VALUE*, and the null value as *PASSING*.

4) Replace GR 5)b)i)2) Otherwise,

Case:

a) If the declared type of TV_i is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with TV_i as *TARGET*, SV_i as *VALUE*, and the <XML passing mechanism> of TV_i as *PASSING*.

b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with TV_i as *TARGET*, SV_i as *VALUE*, and the null value as *PASSING*.

Conformance Rules

No additional Conformance Rules.

14.2 <select statement: single row>

This Subclause modifies Subclause 14.7, "<select statement: single row>", in ISO/IEC 9075-2.

Function

Retrieve values from a specified row of a table.

Format

```
<select target list> ::=
  <target specification 1> [ { <comma> <target specification 1> }... ]
```

Syntax Rules

1) Insert this SR If a <target specification 1> *TS1* contains an <XML passing mechanism>, then:

   a) The declared type of the <target specification> *TS2* contained in *TS1* shall be an XML type.

   b) *TS2* shall not be a <host parameter specification>, an <embedded variable specification>, or a <dynamic parameter specification>.

2) Insert this SR If the declared type of the <target specification> *TS2* contained in a <target specification 1> *TS1* is an XML type, *TS2* is not a <host parameter specification>, an <embedded variable specification>, or a <dynamic parameter specification>, and *TS1* does not contain an <XML passing mechanism>, then it is implementation-defined whether an <XML passing mechanism> of BY REF or BY VALUE is implicit.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 4)a)i) If *TS* is an <SQL parameter reference>, then

   Case:

   a) If the declared type of *TS* is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with *TS* as *TARGET*, the current row as *VALUE*, and the <XML passing mechanism> of *TS* as *PASSING*.

   b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with *TS* as *TARGET*, the current row as *VALUE*, and the null value as *PASSING*.

2) Replace GR 4)b)ii)1)B)V)1)b) The *I*-th element of *A* is set to

   Case:

a) If *EDT* is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with the *I*-th element of *A* as *TARGET*, SL_i as *VALUE*, and the <XML passing mechanism> of TS_i as *PASSING*.

b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with the *I*-th element of *A* as *TARGET*, SL_i as *VALUE*, and the null value as *PASSING*.

3) Replace GR 4)b)ii)1)B)V)2)c) The *I*-th element of *A* is set to

Case:

a) If *EDT* is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with the *I*-th element of *A* as *TARGET*, SL_i as *VALUE*, and the <XML passing mechanism> of TS_i as *PASSING*.

b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with the *I*-th element of *A* as *TARGET*, SL_i as *VALUE*, and the null value as *PASSING*.

4) Replace GR 4)b)ii)2) Otherwise,

Case:

a) If the declared type of TS_i is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with TS_i as *TARGET*, SL_i as *VALUE*, and the <XML passing mechanism> of TS_i as *PASSING*.

b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with TS_i as *TARGET*, SL_i as *VALUE*, and the null value as *PASSING*.

Conformance Rules

No additional Conformance Rules.

14.3 <delete statement: searched>

This Subclause modifies Subclause 14.9, "<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>
      [ FOR PORTION OF <application time period name>
        FROM <point in time 1> TO <point in time 2> ]
      [ [ AS ] <correlation name> ]
      [ 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>.

14.4 <insert statement>

This Subclause modifies Subclause 14.11, "<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>.

14.5 <merge statement>

This Subclause modifies Subclause 14.12, "<merge statement>", in ISO/IEC 9075-2.

Function

Conditionally update and/or delete rows of a table and/or insert new rows into a table.

Format

```
<merge statement> ::=
  MERGE [ WITH <XML lexically scoped options> ] INTO <target table>
      [ [ AS ] <merge 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>.

14.6 <update statement: positioned>

This Subclause modifies Subclause 14.13, "<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> [ [ AS ] <correlation name> ]
      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>.

14.7 <update statement: searched>

This Subclause modifies Subclause 14.14, "<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>
      [ FOR PORTION OF <application time period name>
        FROM <point in time 1> TO <point in time 2> ]
      [ [ AS ] <correlation name> ]
      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>.

(Blank page)

15 Control statements

This Clause modifies Clause 15, "Control statements", in ISO/IEC 9075-4.

15.1 <compound statement>

This Subclause modifies Subclause 15.1, "<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 <assignment statement>

This Subclause modifies Subclause 15.5, "<assignment statement>", in ISO/IEC 9075-4.

Function

Assign a value to an SQL variable, SQL parameter, host parameter, or host variable.

Format

```
<singleton variable assignment> ::=
  SET <assignment target> <equals operator> <assignment source>
      [ <XML passing mechanism> ]
```

Syntax Rules

1) Insert this SR If a <singleton variable assignment> *SVA* contains an <XML passing mechanism>, then:

   a) The declared type of the <assignment target> *AT* contained in *SVA* shall be an XML type.

   b) *AT* shall not be a <host parameter specification>, an <embedded variable specification>, or a <dynamic parameter specification>.

2) Insert this SR If the declared type of the <assignment target> *AT* contained in a <singleton variable assignment> *SVA* is an XML type, *AT* is not a <host parameter specification>, an <embedded variable specification>, or a <dynamic parameter specification>, and *SVA* does not contain an <XML passing mechanism>, then it is implementation-defined whether an <XML passing mechanism> of BY REF or BY VALUE is implicit.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 1) If <assignment target> is a <target specification> that is a <column reference> *T*, an <SQL variable reference> to an SQL variable *T*, or an <SQL parameter reference> to an SQL parameter *T* of an SQL-invoked routine, then

   Case:

   a) If the declared type of *T* is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with *T* as *TARGET*, the value of <assignment source> as *VALUE*, and the <XML passing mechanism> as *PASSING*.

   b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with *T* as *TARGET*, the value of <assignment source> as *VALUE*, and the null value as *PASSING*.

2) Replace GR 4)b) Otherwise,

Case:

a) If the declared type of *FT* is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with *FT* as *TARGET*, the value of <assignment source> as *VALUE*, and the <XML passing mechanism> as *PASSING*.

b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with *FT* as *TARGET*, the value of <assignment source> as *VALUE*, and the null value as *PASSING*.

3) Replace GR 5)b)v)1)B) The *I*-th element of *A* is set to

Case:

a) If *EDT* is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with the *I*-th element of *A* as *TARGET*, the value of <assignment source> as *VALUE*, and the <XML passing mechanism> as *PASSING*.

b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with the *I*-th element of *A* as *TARGET*, the value of <assignment source> as *VALUE*, and the null value as *PASSING*.

4) Replace GR 5)b)v)2)C) The *I*-th element of *A* is set to

Case:

a) If *EDT* is an XML type, then the General Rules of Subclause 10.2, "Store assignment", are applied with the *I*-th element of *A* as *TARGET*, the value of <assignment source> as *VALUE*, and the <XML passing mechanism> as *PASSING*.

b) Otherwise, the General Rules of Subclause 10.2, "Store assignment", are applied with the *I*-th element of *A* as *TARGET*, the value of <assignment source> as *VALUE*, and the null value as *PASSING*.

Conformance Rules

No additional Conformance Rules.

16 Session management

This Clause modifies Clause 19, "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 20, "Dynamic SQL", in ISO/IEC 9075-2.

17.1 Description of SQL descriptor areas

This Subclause modifies Subclause 20.1, "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 28, "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 20.11, "<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) It is implementation-defined whether *XWO* is STRIP WHITESPACE or PRESERVE WHITESPACE.

   c) If the <XML parse>

      ```
      XMLPARSE (XO SV XWO)
      ```

      does not conform to the Syntax Rules of Subclause 6.16, "<XML parse>", then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

   d) The <XML parse>

      ```
      XMLPARSE (XO SV XWO)
      ```

      is effectively performed and is the value of the *i*-th input dynamic parameter.

   NOTE 98 — The effective performance of the <XML parse> includes the raising of exceptions as specified in the General Rules of that Subclause.

Conformance Rules

No additional Conformance Rules.

17.3 <output using clause>

This Subclause modifies Subclause 20.12, "<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 5)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) It is implementation-defined whether *XDO* is INCLUDING XMLDECLARATION or EXCLUDING XMLDECLARATION.

   c) 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*.

   d) If the <string value function>

      ```
      XMLSERIALIZE (XO SV AS TDT XDO)
      ```

      does not conform to the Syntax Rules of Subclause 6.8, "<string value function>", then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

   e) The <string value function>

      ```
      BOM || XMLSERIALIZE (XO SV AS TDT XDO)
      ```

      is effectively performed and is the value *TV* of the *i*-th <target specification>.

      NOTE 99 — The effective performance of the <XML character string serialization> includes the raising of exceptions as specified in the General Rules of that Subclause.

Conformance Rules

No additional Conformance Rules.

17.4 <prepare statement>

This Subclause modifies Subclause 20.7, "<prepare statement>", in ISO/IEC 9075-2.

Function

Prepare a statement for execution.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 5)a)xxxviii) If *DP* is an <XML value expression> simply contained in <XML character string serialization>, <XML binary string serialization>, <XML concatenation>, <XML document>, or <XML validate>, then *DT* is an implementation-defined XML type.

2) Insert after GR 5)a)xxxviii) If *DP* is an <row value predicand> simply contained in <XML content predicate>, <XML document predicate>, or <XML valid predicate>, then *DT* is an implementation-defined XML type.

3) Insert after GR 5)a)xxxviii) If *DP* is a <character value expression> simply contained in <XML comment>, <XML PI>, or <XML text>, then *DT* is CHARACTER VARYING (*ML*).

4) Insert after Note 730

NOTE 100 — The following "part 2" predicates do not have any value expressions with declared type information: <XML content predicate part 2>, <XML document predicate part 2>, <XML valid predicate part 2>.

Conformance Rules

No additional Conformance Rules.

(Blank page)

18 Embedded SQL

This Clause modifies Clause 21, "Embedded SQL", in ISO/IEC 9075-2.

18.1 <embedded SQL host program>

This Subclause modifies Subclause 21.1, "<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 before SR 22)h)ii)2) If *EVN* identifies an XML VARCHAR host variable, an XML CLOB host variable, or an XML BLOB host variable, then:

   a) Let *XO* be the XML option of *EVN*.

   b) Let *XWO* be the XML whitespace option of *EVN*.

   c) *EVN* is replaced by

      ```
      XMLPARSE (XO PN XWO)
      ```

2) Insert before SR 22)l)i)7) For each HVN_i , 1 (one) ≤ i ≤ n, that identifies some HV_i that is either an XML VARCHAR host variable, an XML CLOB host variable, or an XML BLOB host variable, the Syntax Rules of Subclause 9.8, "Host parameter mode determination", in ISO/IEC 9075-2, are applied with the PD_i corresponding to HVN_i as *HOST PARAM DECL* and *ES* as *SQL PROC STMT*, 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 $XHVI_j$.

iv) Let $XSPWI_j$ be the XML whitespace option of $XHVI_j$.

v) It is implementation-defined whether $XDOI_j$ is INCLUDING XMLDECLARATION or EXCLUDING XMLDECLARATION.

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 $XSPWO_k$ be the XML whitespace option of $XHVO_k$.

v) It is implementation-defined whether $XDOO_k$ is INCLUDING XMLDECLARATION or EXCLUDING XMLDECLARATION.

vi) Let XLO_k be the length of $XHVO_k$.

vii) If *XHVOk* is an XML VARCHAR host variable or XML CLOB host variable, then let $XCSO_k$ be the character set of $XHVO_k$.

viii) Case:

1) If $XHVO_k$ is an XML VARCHAR host variable, then let XTO_k be

```
VARCHAR(XLOk) CHARACTER SET XCSOk
```

2) If $XHVO_k$ is an XML BLOB host variable, then let XTO_k be

```
BLOB(XLOk)
```

3) Otherwise, let XTO_k be

```
CLOB(XLOk) CHARACTER SET XCSOk
```

ix) 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 *XSPWIO*$_l$ be the XML whitespace option of *XHVIO*$_l$.

v) It is implementation-defined whether *XDOIO*$_l$ is INCLUDING XMLDECLARATION or EXCLUDING XMLDECLARATION.

vi) Let *XLIO*$_l$ be the length of *XHVIO*$_l$.

vii) If *XHVIO*$_l$ is an XML VARCHAR host variable or XML CLOB host variable, then let *XCSIO*$_l$ be the character set of *XHVIO*$_l$.

viii) Case:

1) If *XHVIO*$_l$ is an XML VARCHAR host variable, then let *XTIO*$_l$ be

```
VARCHAR(XLIOl) CHARACTER SET XCSIOl
```

2) If *XHVIO*$_l$ is an XML BLOB host variable, then let *XTIO*$_l$ be

```
BLOB(XLIOl)
```

3) Otherwise, let *XTIO*$_l$ be

```
CLOB(XLIOl) CHARACTER SET XCSIOl
```

ix) 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*.

3) Insert before SR 22)l)i)7)B) If *HV*$_i$ identifies an XML VARCHAR host variable, an XML CLOB host variable, or an XML BLOB 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*$_l$.

4) Replace SR 22)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 XSPWI1);
    ...
    SET XVId = XMLPARSE (XDOCId XPNId XSPWId);
    SET SVIO1 = TSION1 (CAST (PNIO1 AS TTIO1));
    ...
    SET SVIOc = TSIONc (CAST (PNIOc AS TTIOc));
    SET XVIO1 = XMLPARSE (XDOCIO1 XPNIO1 XSPWIO1);
    ...
    SET XVIOf = XMLPARSE (XDOCIOf XPNIOf XSPWIOf);
    NES;
    SET PNO1 = CAST ( FSON1 (SVO1) AS TSO1);
    ...
    SET PNOb = CAST ( FSONb (SVOb) AS TSOb);
    SET XPNO1 = OBOM1 || XMLSERIALIZE ( XDOCO1 XVO1 AS XTO1 XDOO1 );
    ...
    SET XPNOe = OBOMe || XMLSERIALIZE ( XDOCOe XVOe AS XTOe XDOOe );
    SET PNIO1 = CAST ( FSION1 (SVIO1) AS TSIO1);
    ...
    SET PNIOc = CAST ( FSIONc (SVIOc) AS TSIOc);
    SET XPNIO1 = IOBOM1 || XMLSERIALIZE ( XDOCIO1 XVIO1 AS XTIO1 XDOIO1 );
    ...
    SET XPNIOf = IOBOMf || XMLSERIALIZE ( XDOCIOf XVIOf AS XTIOf XDOIOf );
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 21.3, "<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 BLOB variable>

<Ada XML BLOB variable> ::=
  SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      AS BLOB <left paren> <large object length> <right paren>

<Ada XML CLOB variable> ::=
  SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      AS CLOB <left paren> <character large object length> <right paren>
      [ CHARACTER SET [ IS ] <character set specification> ]
```

Syntax Rules

1) Insert after SR 4)c)xi) If *ATS* is <Ada XML CLOB variable>, then the <host parameter data type> of *HV* is CHARACTER LARGE OBJECT with maximum length specified by the <character large object length> contained in *ATS* and character set specified by the <character set specification> contained in *ATS*. If <character set specification> is not specified, then the character set is implementation-defined.

2) Insert after SR 4)c)xi) If *ATS* is <Ada XML BLOB variable>, then the <host parameter data type> of *HV* is BINARY LARGE OBJECT with maximum length specified by the <large object length> contained in *ATS*.

3) Insert after SR 5)f) The syntax

```
SQL TYPE IS XML XO XWO 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..LL);
END RECORD;
```

in any <Ada XML CLOB variable>, where:

a) *NV* is the numeric value of *L* as specified in Subclause 6.1, "<data type>".

b) The value of *LL* is *NV*k*.

c) ***XO*** is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

d) ***XWO*** is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

e) *CS* is a <character set name> as specified in Subclause 5.2, "Names and identifiers".

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*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the *XML option of XML CLOB host variable*; otherwise *XO* is the *XML option of XML CLOB host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML CLOB host variable*; otherwise, *XWO* is the *XML whitespace option of the XML CLOB host variable*.

NOTE 101 — The length of the XML CLOB host variable contains an implicit or explicit <char length units>.

4) Insert after SR 5)f) The syntax

```
SQL TYPE IS XML XO XWO AS BLOB (L)
```

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 BLOB variable>, where:

a) *L* is the numeric value of <large object length> as specified in Subclause 6.1, "<data type>".

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) XWO is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

HVN is an *XML BLOB host variable*. *L* is the *length of XML BLOB host variable*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the *XML option of XML BLOB host variable*; otherwise *XO* is the *XML option of XML BLOB host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML BLOB host variable*; otherwise, *XWO* is the *XML whitespace option of the XML BLOB 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.

4) Insert this CR Without Feature X112, “Host language support for XML: BLOB mapping”, conforming SQL language shall not contain an <Ada XML BLOB variable>.

5) Insert this CR Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <Ada XML BLOB variable> shall not immediately contain a <document or content> that is CONTENT.

6) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <Ada XML BLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

7) Insert this CR Without Feature X113, “Host language support for XML: STRIP WHITESPACE option”, in conforming SQL language, <Ada XML CLOB variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

8) Insert this CR Without Feature X113, “Host language support for XML: STRIP WHITESPACE option”, in conforming SQL language, <Ada XML BLOB variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

9) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <Ada XML CLOB variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

10) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <Ada XML BLOB variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

18.3 <embedded SQL C program>

This Subclause modifies Subclause 21.4, "<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 BLOB variable>

<C XML VARCHAR variable> ::=
  SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      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> ]
      [ <XML whitespace option> ]
      AS CLOB <left paren> <character large object length> <right paren>
      [ CHARACTER SET [ IS ] <character set specification> ]
      <C host identifier> [ <C initial value> ]
      [ { <comma> <C host identifier> [ <C initial value> ] }... ]

<C XML BLOB variable> ::=
  SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      AS BLOB <left paren> <large object length> <right paren>
      <C host identifier> [ <C initial value> ]
      [ { <comma> <C host identifier> [ <C initial value> ] }... ]
```

Syntax Rules

1) Insert after SR 4)c)x) If *CVS* contains <C XML VARCHAR variable>, then the <host parameter data type> of *HV* is CHARACTER VARYING, with maximum length specified by the <character length> of the <C array specification>, measured in the units specified by the explicit or implicit <char length units>. The character set is specified by the <character set specification>; if omitted, the character set is implementation-defined. The value in *HV* is terminated by a null character and the position occupied by this null character is included in the maximum length of *HV*. 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 <character length> of the <C array specification> and whose value does not include the terminating null character. The <length> shall be greater than 1 (one).

2) Insert after SR 4)c)x) If *CVS* contains <C XML CLOB variable>, then the <host parameter data type> of *HV* is CHARACTER LARGE OBJECT, with maximum length specified by the <character large object length>, and with character set specified by the <character set specification>. If there is no <character set specification>, then the character set is implementation-defined. 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 with maximum length and character set of the <host parameter data type> of *HV*.

3) Insert after SR 4)c)x) If *CVS* contains <C XML BLOB variable>, then the <host parameter data type> of *HV* is BINARY LARGE OBJECT, with maximum length specified by the <large object length>.

4) 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 XML VARCHAR variable>, or a <C XML CLOB variable>.

5) Insert after SR 5)f) The syntax

```
SQL TYPE IS XML XO XWO AS VARCHAR
  CHARACTER SET IS CS hvn[L]
```

for a given <C host identifier> *hvn* shall be replaced by:

```
char hvn [LL]
```

in any <C XML VARCHAR variable>, where:

a) *NV* is the numeric value of *L* as specified in Subclause 6.1, "<data type>" and the value of *LL* is *NV*k*.

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) *XWO* is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

d) *CS* is a <character set name> as specified in Subclause 5.2, "Names and identifiers".

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*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the XML option of XML VARCHAR host variable; otherwise *XO* is the *XML option of XML VARCHAR host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML VARCHAR host variable*; otherwise *XWO* is the *XML whitespace option of the XML VARCHAR host variable*.

NOTE 102 — The length of the XML VARCHAR host variable contains an implicit or explicit <char length units>.

6) Insert after SR 5)f) The syntax

```
SQL TYPE IS XML XO XWO 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[LL];
} hvn
```

in any <C XML CLOB variable>, where:

a) *NV* is the numeric value of *L* as specified in Subclause 6.1, "<data type>", and the value of *LL* is *NV***k*.

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) *XWO* is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

d) *CS* is a <character set name> as specified in Subclause 5.2, "Names and identifiers".

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*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the XML option of XML CLOB host variable; otherwise *XO* is the *XML option of XML CLOB host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML CLOB host variable*; otherwise *XWO* is the *XML whitespace option of the XML CLOB host variable*.

NOTE 103 — The length of the XML CLOB host variable contains an implicit or explicit <char length units>.

7) Insert after SR 5)f) The syntax

```
SQL TYPE IS XML XO XWO AS BLOB(L)
```

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 BLOB variable>, where:

a) *L* is the numeric value of <large object length> as specified in Subclause 6.1, "<data type>".

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) *XWO* is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

hvn is an *XML BLOB host variable*. *L* is the *length of XML BLOB host variable*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the XML option of XML BLOB host variable; otherwise *XO* is the *XML option of XML BLOB host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML BLOB host variable*; otherwise *XWO* is the *XML whitespace option of the XML BLOB host variable*.

8) Replace SR 7) In a <C variable definition>, the words "VARCHAR", "CHARACTER", "SET", "IS", "VARYING", "BINARY", "VARBINARY" "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>").

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.

5) Insert this CR Without Feature X112, "Host language support for XML: BLOB mapping", conforming SQL language shall not contain a <C XML BLOB variable>.

6) Insert this CR Without Feature X100, "Host language support for XML: CONTENT option", in conforming SQL language, <C XML BLOB variable> shall not immediately contain a <document or content> that is CONTENT.

7) Insert this CR Without Feature X101, "Host language support for XML: DOCUMENT option", in conforming SQL language, <C XML BLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

8) Insert this CR Without Feature X113, "Host language support for XML: STRIP WHITESPACE option", in conforming SQL language, <C XML CLOB variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

9) Insert this CR Without Feature X113, "Host language support for XML: STRIP WHITESPACE option", in conforming SQL language, <C XML BLOB variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

10) Insert this CR Without Feature X113, "Host language support for XML: STRIP WHITESPACE option", in conforming SQL language, <C XML VARCHAR variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

11) Insert this CR Without Feature X114, "Host language support for XML: PRESERVE WHITESPACE option", in conforming SQL language, <C XML CLOB variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

12) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <C XML BLOB variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

13) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <C XML VARCHAR variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

18.4 <embedded SQL COBOL program>

This Subclause modifies Subclause 21.5, "<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 BLOB variable>

<COBOL XML BLOB variable> ::=
  [ USAGE [ IS ] ] SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      AS BLOB <left paren> <large object length> <right paren>

<COBOL XML CLOB variable> ::=
  [ USAGE [ IS ] ] SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      AS CLOB <left paren> <character large object length> <right paren>
      [ CHARACTER SET [ IS ] <character set specification> ]
```

Syntax Rules

1) Insert after SR 4)d)x) If *CTS* contains <COBOL XML CLOB variable>, then the <host parameter data type> of *HV* is CHARACTER LARGE OBJECT with the maximum length specified by <character large object 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.

2) Insert after SR 4)d)x) If *CTS* contains <COBOL XML BLOB variable>, then the <host parameter data type> of *HV* is BINARY LARGE OBJECT with maximum length specified by <large object length> contained in *CTS*.

3) Insert after SR 5)d) The syntax

```
SQL TYPE IS XML XO XWO 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(LL).
```

in any <COBOL XML CLOB variable>, where:

a) *NV* is the numeric value of *L* as specified in Subclause 6.1, "<data type>" and the value of *LL* is *NV***k*.

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) *XWO* is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

d) *CS* is a <character set name> as specified in Subclause 5.2, "Names and identifiers".

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*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the XML option of XML CLOB host variable; otherwise *XO* is the *XML option of XML CLOB host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML CLOB host variable*; otherwise *XWO* is the *XML whitespace option of the XML CLOB host variable*.

NOTE 104 — The length of the XML CLOB host variable contains an implicit or explicit <char length units>.

4) Insert after SR 5)d) The syntax

```
SQL TYPE IS XML XO XWO AS BLOB ( L )
```

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 BLOB variable>, where:

a) *L* is the numeric value of <large object length> as specified in Subclause 6.1, "<data type>".

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) *XWO* is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

HVN is an *XML BLOB host variable*. *L* is the *length of XML BLOB host variable*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the XML option of XML BLOB host variable; otherwise *XO* is the *XML option of XML BLOB host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML BLOB host variable*; otherwise *XWO* is the *XML whitespace option of the XML BLOB 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 <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.

4) Insert this CR Without Feature X112, “Host language support for XML: BLOB mapping”, conforming SQL language shall not contain a <COBOL XML BLOB variable>.

5) Insert this CR Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <COBOL XML BLOB variable> shall not immediately contain a <document or content> that is CONTENT.

6) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <COBOL XML BLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

7) Insert this CR Without Feature X113, “Host language support for XML: STRIP WHITESPACE option”, in conforming SQL language, <COBOL XML CLOB variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

8) Insert this CR Without Feature X113, “Host language support for XML: STRIP WHITESPACE option”, in conforming SQL language, <COBOL XML BLOB variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

9) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <COBOL XML CLOB variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

10) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <COBOL XML BLOB variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

18.5 <embedded SQL Fortran program>

This Subclause modifies Subclause 21.6, "<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 BLOB variable>

<Fortran XML BLOB variable> ::=
  SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      AS BLOB <left paren> <large object length> <right paren>

<Fortran XML CLOB variable> ::=
  SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      AS CLOB <left paren> <character large object length> <right paren>
      [ CHARACTER SET [ IS ] <character set specification> ]
```

Syntax Rules

1) Insert after SR 5)g)xi) If *FTS* contains <Fortran XML CLOB variable>, then the <host parameter data type> of *HV* is CHARACTER LARGE OBJECT with the maximum length specified by the <character large object 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.

2) Insert after SR 5)g)xi) If *FTS* contains <Fortran XML BLOB variable>, then the <host parameter data type> of *HV* is BINARY LARGE OBJECT with the maximum length specified by the <large object length> contained in *FTS*.

3) Insert after SR 6)d) The syntax

```
SQL TYPE IS XML XO XWO AS CLOB ( L )
  CHARACTER SET IS CS
```

for a given <Fortran host identifier> *HVN* shall be replaced by

```
CHARACTER HVN (L+8)
INTEGER*4 HVN_RESERVED
INTEGER*4 HVN_LENGTH
CHARACTER HVN_DATA * LL
EQUIVALENCE (HVN(1), HVN_RESERVED)
EQUIVALENCE (HVN(5), HVN_LENGTH)
EQUIVALENCE (HVN(9), HVN_DATA)
```

in any <Fortran XML CLOB variable>, where:

a) *NV* is the numeric value of *L* as specified in Subclause 6.1, "<data type>" and the value of *LL* is *NV***k*.

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) *XWO* is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

d) *CS* is a <character set name> as specified in Subclause 5.2, "Names and identifiers".

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*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the XML option of XML CLOB host variable; otherwise *XO* is the *XML option of XML CLOB host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML CLOB host variable*; otherwise *XWO* is the *XML whitespace option of the XML CLOB host variable*.

NOTE 105 — The length of the XML CLOB host variable contains an implicit or explicit <char length units>.

4) Insert after SR 6)d) The syntax

```
SQL TYPE IS XML XO XWO AS BLOB ( L )
```

for a given <Fortran host identifier> *HVN* shall be replaced by

```
CHARACTER HVN (L+8)
INTEGER*4 HVN_RESERVED
INTEGER*4 HVN_LENGTH
CHARACTER HVN_DATA * L
EQUIVALENCE (HVN(1), HVN_RESERVED)
EQUIVALENCE (HVN(5), HVN_LENGTH)
EQUIVALENCE (HVN(9), HVN_DATA)
```

in any <Fortran XML BLOB variable>, where:

a) *L* is the numeric value of <large object length> as specified in Subclause 6.1, "<data type>".

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) *XWO* is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

HVN is an *XML BLOB host variable*. *L* is the *length of XML BLOB host variable*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the XML option of XML BLOB host variable; otherwise *XO* is the *XML option of XML BLOB host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML BLOB host variable*; otherwise *XWO* is the *XML whitespace option of the XML BLOB 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 <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.

4) Insert this CR Without Feature X112, “Host language support for XML: BLOB mapping”, conforming SQL language shall not contain a <Fortran XML BLOB variable>.

5) Insert this CR Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <Fortran XML BLOB variable> shall not immediately contain a <document or content> that is CONTENT.

6) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <Fortran XML BLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

7) Insert this CR Without Feature X113, “Host language support for XML: STRIP WHITESPACE option”, in conforming SQL language, <Fortran XML CLOB variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

8) Insert this CR Without Feature X113, “Host language support for XML: STRIP WHITESPACE option”, in conforming SQL language, <Fortran XML BLOB variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

9) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <Fortran XML CLOB variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

10) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <Fortran XML BLOB variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

18.6 <embedded SQL Pascal program>

This Subclause modifies Subclause 21.8, "<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 BLOB variable>

<Pascal XML CLOB variable> ::=
  SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      AS CLOB <left paren> <character large object length> <right paren>
      [ CHARACTER SET [ IS ] <character set specification> ]

<Pascal XML BLOB variable> ::=
  SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      AS BLOB <left paren> <large object length> <right paren>
```

Syntax Rules

1) Insert after SR 4)f)x) If *PTS* is <Pascal XML CLOB variable>, then the <host parameter data type> of *HV* is CHARACTER LARGE OBJECT with maximum length specified by the <character large object length> contained in *PTS* and character set specified by the <character set specification> contained in *PTS*. If <character set specification> is not specified, then the character set is implementation-defined.

2) Insert after SR 4)f)x) If *PTS* is <Pascal XML BLOB variable>, then the <host parameter data type> of *HV* is BINARY LARGE OBJECT with maximum length specified by the <large object length> contained in *PTS*.

3) 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>.

4) Insert after SR 5)f) The syntax

```
SQL TYPE IS XML XO XWO 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..LL ] OF CHAR ;
END ;
```

in any <Pascal XML CLOB variable>, where:

a) *NV* is the numeric value of *L* as specified in Subclause 6.1, "<data type>", and the value of *LL* is *NV***k*.

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) *XWO* is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

d) *CS* is a <character set name> as specified in Subclause 5.2, "Names and identifiers".

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*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the XML option of XML CLOB host variable; otherwise *XO* is the *XML option of XML CLOB host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML CLOB host variable*; otherwise *XWO* is the *XML whitespace option of the XML CLOB host variable*.

NOTE 106 — The length of the XML CLOB host variable contains an implicit or explicit <char length units>.

5) Insert after SR 5)f) The syntax

```
SQL TYPE IS XML XO XWO AS BLOB ( L )
```

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 BLOB variable>, where:

a) *L* is the numeric value of <large object length> as specified in Subclause 6.1, "<data type>".

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) *XWO* is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

HVN is an *XML BLOB host variable*. *L* is the *length of XML BLOB host variable*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the XML option of XML BLOB host variable; otherwise *XO* is the *XML option of XML BLOB host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML BLOB host variable*; otherwise *XWO* is the *XML whitespace option of the XML BLOB 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.

4) Insert this CR Without Feature X112, “Host language support for XML: BLOB mapping”, conforming SQL language shall not contain a <Pascal XML BLOB variable>.

5) Insert this CR Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <Pascal XML BLOB variable> shall not immediately contain a <document or content> that is CONTENT.

6) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <Pascal XML BLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

7) Insert this CR Without Feature X113, “Host language support for XML: STRIP WHITESPACE option”, in conforming SQL language, <Pascal XML CLOB variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

8) Insert this CR Without Feature X113, “Host language support for XML: STRIP WHITESPACE option”, in conforming SQL language, <Pascal XML BLOB variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

9) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <Pascal XML CLOB variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

10) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <Pascal XML BLOB variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

18.7 <embedded SQL PL/I program>

This Subclause modifies Subclause 21.9, "<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 BLOB variable>

<PL/I XML VARCHAR variable> ::=
  SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      AS VARCHAR <left paren> <character length> <right paren>
      [ CHARACTER SET [ IS ] <character set specification> ]

<PL/I XML CLOB variable> ::=
  SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      AS CLOB <left paren> <character large object length> <right paren>
      [ CHARACTER SET [ IS ] <character set specification> ]

<PL/I XML BLOB variable> ::=
  SQL TYPE IS XML [ <document or content> ]
      [ <XML whitespace option> ]
      AS BLOB <left paren> <large object length> <right paren>
```

Syntax Rules

1) Insert after SR 4)e)xi) If *PTS* contains <PL/I XML VARCHAR variable>, then the <host parameter data type> of *HV* is CHARACTER VARYING with maximum length specified by the <character length> contained in *PTS* and character set is specified by the <character set specification> contained in *PTS*. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

2) Insert after SR 4)e)xi) If *PTS* contains <PL/I XML CLOB variable>, then the <host parameter data type> of *HV* is CHARACTER LARGE OBJECT with maximum length specified by the <character large object length> contained in *PTS* and with character set set specified by the <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

3) Insert after SR 4)e)xi) If *PTS* contains <PL/I XML BLOB variable>, then the <host parameter data type> of *HV* is BINARY LARGE OBJECT with maximum length specified by the <large object length> contained in *PTS*.

4) Insert after SR 5)d) The syntax

```
SQL TYPE IS XML XO XWO AS VARCHAR (L)
  CHARACTER SET IS CS
```

for a given <PL/I host identifier> *HVN* shall be replaced by:

```
CHARACTER VARYING (LL)
```

in any <PL/I XML VARCHAR variable>, where:

a) *NV* is the numeric value of *L* as specified in Subclause 6.1, "<data type>", and the value of *LL* is *NV***k*.

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) *XWO* is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

d) *CS* is a <character set name> as specified in Subclause 5.2, "Names and identifiers".

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*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the XML option of XML VARCHAR host variable; otherwise *XO* is the *XML option of XML VARCHAR host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML VARCHAR host variable*; otherwise *XWO* is the *XML whitespace option of the XML VARCHAR host variable*.

NOTE 107 — The length of the XML VARCHAR host variable contains an implicit or explicit <char length units>.

5) Insert after SR 5)d) The syntax

```
SQL TYPE IS XML XO XWO 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 ( LL );
```

in any <PL/I XML CLOB variable>, where:

a) *NV* is the numeric value of *L* as specified in Subclause 6.1, "<data type>", and the value of *LL* is *NV***k*.

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) *XWO* is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

d) *CS* is a <character set name> as specified in Subclause 5.2, "Names and identifiers".

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*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the XML option of XML CLOB host variable; otherwise *XO* is the *XML option of XML CLOB host variable*. If *XWO* is the zero-length string, then it is implementation-defined

whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML CLOB host variable*; otherwise *XWO* is the *XML whitespace option of the XML CLOB host variable*.

NOTE 108 — The length of the XML CLOB host variable contains an implicit or explicit <char length units>.

6) Insert after SR 5)d) The syntax

```
SQL TYPE IS XML XO XWO AS BLOB ( L )
```

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 ( L );
```

in any <PL/I XML BLOB variable>, where:

a) *L* is the numeric value of <large object length> as specified in Subclause 6.1, "<data type>".

b) *XO* is either <document or content> as specified in Subclause 6.8, "<string value function>", or the zero-length string.

c) *XWO* is either <XML whitespace option> as specified in Subclause 6.16, "<XML parse>", or the zero-length string.

HVN is an *XML BLOB host variable*. *L* is the *length of XML BLOB host variable*. If *XO* is the zero-length string, then it is implementation-defined whether DOCUMENT or CONTENT is the XML option of XML BLOB host variable; otherwise *XO* is the *XML option of XML BLOB host variable*. If *XWO* is the zero-length string, then it is implementation-defined whether STRIP WHITESPACE or PRESERVE WHITESPACE is the *XML whitespace option of the XML BLOB host variable*; otherwise *XWO* is the *XML whitespace option of the XML BLOB host variable*.

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.

5) Insert this CR Without Feature X112, “Host language support for XML: BLOB mapping”, conforming SQL language shall not contain a <PL/I XML BLOB variable>.

6) Insert this CR Without Feature X100, “Host language support for XML: CONTENT option”, in conforming SQL language, <PL/I XML BLOB variable> shall not immediately contain a <document or content> that is CONTENT.

7) Insert this CR Without Feature X101, “Host language support for XML: DOCUMENT option”, in conforming SQL language, <PL/I XML BLOB variable> shall not immediately contain a <document or content> that is DOCUMENT.

8) Insert this CR Without Feature X113, “Host language support for XML: STRIP WHITESPACE option”, in conforming SQL language, <PL/I XML CLOB variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

9) Insert this CR Without Feature X113, “Host language support for XML: STRIP WHITESPACE option”, in conforming SQL language, <PL/I XML BLOB variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

10) Insert this CR Without Feature X113, “Host language support for XML: STRIP WHITESPACE option”, in conforming SQL language, <PL/I XML VARCHAR variable> shall not immediately contain an <XML whitespace option> that is STRIP WHITESPACE.

11) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <PL/I XML CLOB variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

12) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <PL/I XML BLOB variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

13) Insert this CR Without Feature X114, “Host language support for XML: PRESERVE WHITESPACE option”, in conforming SQL language, <PL/I XML VARCHAR variable> shall not immediately contain an <XML whitespace option> that is PRESERVE WHITESPACE.

19 Call-Level Interface specifications

This Clause modifies Clause 5, "Call-Level Interface specifications", in ISO/IEC 9075-3.

19.1 SQL/CLI data type correspondences

This Subclause modifies Subclause 5.20, "SQL/CLI data type correspondences", in ISO/IEC 9075-3.

Function

Replace first paragraph Specify the SQL/CLI data type correspondences for SQL data types and host language types associated with the required parameter mechanisms, as shown in Table 3, "Supported calling conventions of SQL/CLI routines by language", in [ISO9075-3].

Tables

Table 13, "SQL/CLI data type correspondences for Ada", modifies Table 40, "SQL/CLI data type correspondences for Ada", in [ISO9075-3].

Table 13 — SQL/CLI data type correspondences for Ada

SQL Data Type	Ada Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*
XML	*None*

Table 14, "SQL/CLI data type correspondences for C", modifies Table 41, "SQL/CLI data type correspondences for C", in [ISO9075-3].

Table 14 — SQL/CLI data type correspondences for C

SQL Data Type	C Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*
XML	*None*

Table 15, "SQL/CLI data type correspondences for COBOL", modifies Table 42, "SQL/CLI data type correspondences for COBOL", in [ISO9075-3].

Table 15 — SQL/CLI data type correspondences for COBOL

SQL Data Type	COBOL Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*
XML	*None*

Table 16, "SQL/CLI data type correspondences for Fortran", modifies Table 43, "SQL/CLI data type correspondences for Fortran", in [ISO9075-3].

Table 16 — SQL/CLI data type correspondences for Fortran

SQL Data Type	Fortran Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*
XML	*None*

Table 17, "SQL/CLI data type correspondences for M", modifies Table 44, "SQL/CLI data type correspondences for M", in [ISO9075-3].

Table 17 — SQL/CLI data type correspondences for M

SQL Data Type	MUMPS Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*
XML	*None*

Table 18, "SQL/CLI data type correspondences for Pascal", modifies Table 45, "SQL/CLI data type correspondences for Pascal", in [ISO9075-3].

Table 18 — SQL/CLI data type correspondences for Pascal

SQL Data Type	Pascal Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*
XML	*None*

Table 19, "SQL/CLI data type correspondences for PL/I", modifies Table 46, "SQL/CLI data type correspondences for PL/I", in [ISO9075-3].

Table 19 — SQL/CLI data type correspondences for PL/I

SQL Data Type	PL/I Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*
XML	*None*

(Blank page)

20 Diagnostics management

This Clause modifies Clause 23, "Diagnostics management", in ISO/IEC 9075-2.

20.1 <get diagnostics statement>

This Subclause modifies Subclause 23.1, "<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 37, "SQL-statement codes"

Table 20 — 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.

(Blank page)

21 Information Schema

This Clause modifies Clause 5, "Information Schema", in ISO/IEC 9075-11.

21.1 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 109 — 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.

21.2 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.

21.3 ATTRIBUTES view

This Subclause modifies Subclause 5.11, "ATTRIBUTES view", in ISO/IEC 9075-11.

Function

Identify the attributes of user-defined types defined in this catalog that are accessible to a given user or role.

Definition

Add the following columns to the end of outermost select list of the view definition

```
,
D1.XML_PRIMARY_MODIFIER, D1.XML_SECONDARY_MODIFIER,
D1.XML_SCHEMA_CATALOG, D1.XML_SCHEMA_SCHEMA, D1.XML_SCHEMA_NAME,
D1.XML_SCHEMA_NAMESPACE, D1.XML_SCHEMA_ELEMENT
```

Conformance Rules

1) Insert this CR Without Feature X160, "Basic Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.ATTRIBUTES.XML_SCHEMA_CATALOG, INFORMATION_SCHEMA.ATTRIBUTES.XML_SCHEMA_SCHEMA, or INFORMATION_SCHEMA.ATTRIBUTES.XML_SCHEMA_NAME.

2) Insert this CR Without Feature X161, "Advanced Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.ATTRIBUTES.XML_SCHEMA_NAMESPACE or INFORMATION_SCHEMA.ATTRIBUTES.XML_SCHEMA_ELEMENT.

21.4 COLUMNS view

This Subclause modifies Subclause 5.21, "COLUMNS view", in ISO/IEC 9075-11.

Function

Identify the columns of tables defined in this catalog that are accessible to a given user or role.

Definition

Add the following columns to the end of outermost select list of the view definition

```
,
COALESCE (D1.XML_PRIMARY_MODIFIER, D2.XML_PRIMARY_MODIFIER) AS
         XML_PRIMARY_MODIFIER,
COALESCE (D1.XML_SECONDARY_MODIFIER, D2.XML_SECONDARY_MODIFIER) AS
         XML_SECONDARY_MODIFIER,
COALESCE (D1.XML_SCHEMA_CATALOG, D2.XML_SCHEMA_CATALOG) AS
         XML_SCHEMA_CATALOG,
COALESCE (D1.XML_SCHEMA_SCHEMA, D2.XML_SCHEMA_SCHEMA) AS
         XML_SCHEMA_SCHEMA,
COALESCE (D1.XML_SCHEMA_NAME, D2.XML_SCHEMA_NAME) AS
         XML_SCHEMA_NAME,
COALESCE (D1.XML_SCHEMA_NAMESPACE, D2.XML_SCHEMA_NAMESPACE) AS
         XML_SCHEMA_NAMESPACE,
COALESCE (D1.XML_SCHEMA_ELEMENT, D2.XML_SCHEMA_ELEMENT) AS
         XML_SCHEMA_ELEMENT
```

Conformance Rules

1) Insert this CR Without Feature X160, "Basic Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.COLUMNS.XML_SCHEMA_CATALOG, INFORMATION_SCHEMA.COLUMNS.XML_SCHEMA_SCHEMA, or INFORMATION_SCHEMA.COLUMNS.XML_SCHEMA_NAME.

2) Insert this CR Without Feature X161, "Advanced Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.COLUMNS.XML_SCHEMA_NAMESPACE or INFORMATION_SCHEMA.COLUMNS.XML_SCHEMA_ELEMENT.

21.5 DOMAINS view

This Subclause modifies Subclause 5.29, "DOMAINS view", in ISO/IEC 9075-11.

Function

Identify the domains defined in this catalog that are accessible to a given user or role.

Definition

Add the following columns to the end of outermost select list of the view definition

```
,
DTD.XML_PRIMARY_MODIFIER, DTD.XML_SECONDARY_MODIFIER,
DTD.XML_SCHEMA_CATALOG, DTD.XML_SCHEMA_SCHEMA, DTD.XML_SCHEMA_NAME,
DTD.XML_SCHEMA_NAMESPACE, DTD.XML_SCHEMA_ELEMENT
```

Conformance Rules

1) Insert this CR Without Feature X160, "Basic Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.DOMAINS.XML_SCHEMA_CATALOG, INFORMATION_SCHEMA.DOMAINS.XML_SCHEMA_SCHEMA, or INFORMATION_SCHEMA.DOMAINS.XML_SCHEMA_NAME.

2) Insert this CR Without Feature X161, "Advanced Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.DOMAINS.XML_SCHEMA_NAMESPACE or INFORMATION_SCHEMA.DOMAINS.XML_SCHEMA_ELEMENT.

21.6 ELEMENT_TYPES view

This Subclause modifies Subclause 5.30, "ELEMENT_TYPES view", in ISO/IEC 9075-11.

Function

Identify the collection element types defined in this catalog that are accessible to a given user or role.

Definition

Add the following columns to the end of outermost select list of the view definition

```
,
DTD.XML_PRIMARY_MODIFIER, DTD.XML_SECONDARY_MODIFIER,
DTD.XML_SCHEMA_CATALOG, DTD.XML_SCHEMA_SCHEMA, DTD.XML_SCHEMA_NAME,
DTD.XML_SCHEMA_NAMESPACE, DTD.XML_SCHEMA_ELEMENT
```

Conformance Rules

1) Insert this CR Without Feature X160, "Basic Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.ELEMENT_TYPES.XML_SCHEMA_CATALOG, INFORMATION_SCHEMA.ELEMENT_TYPES.XML_SCHEMA_SCHEMA, or INFORMATION_SCHEMA.ELEMENT_TYPES.XML_SCHEMA_NAME.

2) Insert this CR Without Feature X161, "Advanced Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.ELEMENT_TYPES.XML_SCHEMA_NAMESPACE or INFORMATION_SCHEMA.ELEMENT_TYPES.XML_SCHEMA_ELEMENT.

21.7 FIELDS view

This Subclause modifies Subclause 5.32, "FIELDS view", in ISO/IEC 9075-11.

Function

Identify the field types defined in this catalog that are accessible to a given user or role.

Definition

Add the following columns to the end of outermost select list of the view definition

```
,
DTD.XML_PRIMARY_MODIFIER, DTD.XML_SECONDARY_MODIFIER,
DTD.XML_SCHEMA_CATALOG, DTD.XML_SCHEMA_SCHEMA, DTD.XML_SCHEMA_NAME,
DTD.XML_SCHEMA_NAMESPACE, DTD.XML_SCHEMA_ELEMENT
```

Conformance Rules

1) Insert this CR Without Feature X160, "Basic Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.FIELDS.XML_SCHEMA_CATALOG, INFORMATION_SCHEMA.FIELDS.XML_SCHEMA_SCHEMA, or INFORMATION_SCHEMA.FIELDS.XML_SCHEMA_NAME.

2) Insert this CR Without Feature X161, "Advanced Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.FIELDS.XML_SCHEMA_NAMESPACE or INFORMATION_SCHEMA.FIELDS.XML_SCHEMA_ELEMENT.

21.8 METHOD_SPECIFICATION_PARAMETERS view

This Subclause modifies Subclause 5.35, "METHOD_SPECIFICATION_PARAMETERS view", in ISO/IEC 9075-11.

Function

Identify the SQL parameters of method specifications described in the METHOD_SPECIFICATIONS view that are accessible to a given user or role.

Definition

Add the following columns to the end of outermost select list of the view definition.

```
,
D.XML_PRIMARY_MODIFIER, D.XML_SECONDARY_MODIFIER,
D.XML_SCHEMA_CATALOG, D.XML_SCHEMA_SCHEMA, D.XML_SCHEMA_NAME,
D.XML_SCHEMA_NAMESPACE, D.XML_SCHEMA_ELEMENT
```

Conformance Rules

1) Insert this CR Without Feature X160, "Basic Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.METHOD_SPECIFICATION_PARAMETERS.XML_SCHEMA_CATALOG, INFORMATION_SCHEMA.METHOD_SPECIFICATION_PARAMETERS.XML_SCHEMA_SCHEMA, or INFORMATION_SCHEMA.METHOD_SPECIFICATION_PARAMETERS.XML_SCHEMA_NAME.

2) Insert this CR Without Feature X161, "Advanced Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.METHOD_SPECIFICATION_PARAMETERS.XML_SCHEMA_NAMESPACE or INFORMATION_SCHEMA.METHOD_SPECIFICATION_PARAMETERS.XML_SCHEMA_ELEMENT.

21.9 METHOD_SPECIFICATIONS view

This Subclause modifies Subclause 5.36, "METHOD_SPECIFICATIONS view", in ISO/IEC 9075-11.

Function

Identify the SQL-invoked methods in the catalog that are accessible to a given user or role.

Definition

Add the following columns to the end of outermost select list of the view definition

```
,
D.XML_PRIMARY_MODIFIER, D.XML_SECONDARY_MODIFIER,
D.XML_SCHEMA_CATALOG, D.XML_SCHEMA_SCHEMA, D.XML_SCHEMA_NAME,
D.XML_SCHEMA_NAMESPACE, D.XML_SCHEMA_ELEMENT,
DT.XML_PRIMARY_MODIFIER AS RESULT_CAST_XML_PRIMARY_MODIFIER,
DT.XML_SECONDARY_MODIFIER AS RESULT_CAST_XML_SECONDARY_MODIFIER,
DT.XML_SCHEMA_CATALOG AS RESULT_CAST_XML_SCHEMA_CATALOG,
DT.XML_SCHEMA_SCHEMA AS RESULT_CAST_XML_SCHEMA_SCHEMA,
DT.XML_SCHEMA_NAME AS RESULT_CAST_XML_SCHEMA_NAME,
DT.XML_SCHEMA_NAMESPACE AS RESULT_CAST_XML_SCHEMA_NAMESPACE,
DT.XML_SCHEMA_ELEMENT AS RESULT_CAST_XML_SCHEMA_ELEMENT
```

Conformance Rules

1) Insert this CR Without Feature X160, "Basic Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.METHOD_SPECIFICATIONS.XML_SCHEMA_CATALOG, INFORMATION_SCHEMA.METHOD_SPECIFICATIONS.XML_SCHEMA_SCHEMA, INFORMATION_SCHEMA.METHOD_SPECIFICATIONS.XML_SCHEMA_NAME, INFORMATION_SCHEMA.METHOD_SPECIFICATIONS.RESULT_CAST_XML_SCHEMA_CATALOG, INFORMATION_SCHEMA.METHOD_SPECIFICATIONS.RESULT_CAST_XML_SCHEMA_SCHEMA, or INFORMATION_SCHEMA.METHOD_SPECIFICATIONS.RESULT_CAST_XML_SCHEMA_NAME.

2) Insert this CR Without Feature X161, "Advanced Information Schema for registered XML Schemas", conforming SQL language shall not reference INFORMATION_SCHEMA.METHOD_SPECIFICATIONS.XML_SCHEMA_NAMESPACE, INFORMATION_SCHEMA.METHOD_SPECIFICATIONS.XML_SCHEMA_ELEMENT, INFORMATION_SCHEMA.METHOD_SPECIFICATIONS.RESULT_CAST_XML_SCHEMA_NAMESPACE, or INFORMATION_SCHEMA.METHOD_SPECIFICATIONS.RESULT_CAST_XML_SCHEMA_ELEMENT.

21.10 PARAMETERS view

This Subclause modifies Subclause 5.37, "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

Add the following columns to the <select list>

```
,
DTD.XML_PRIMARY_MODIFIER, DTD.XML_SECONDARY_MODIFIER,
DTD.XML_SCHEMA_CATALOG, DTD.XML_SCHEMA_SCHEMA, DTD.XML_SCHEMA_NAME,
DTD.XML_SCHEMA_NAMESPACE, DTD.XML_SCHEMA_ELEMENT,
DTD2.DATA_TYPE AS XML_CHAR_TYPE,
DTD2.CHARACTER_MAXIMUM_LENGTH AS XML_CHAR_MAX_LEN,
DTD2.CHARACTER_OCTET_LENGTH AS XML_CHAR_OCT_LEN,
DTD2.CHARACTER_SET_CATALOG AS XML_CHAR_SET_CAT,
DTD2.CHARACTER_SET_SCHEMA AS XML_CHAR_SET_SCH,
DTD2.CHARACTER_SET_NAME AS XML_CHAR_SET_NAME,
DTD2.COLLATION_CATALOG AS XML_CHAR_COLL_CAT,
DTD2.COLLATION_SCHEMA AS XML_CHAR_COLL_SCH,
DTD2.COLLATION_NAME AS XML_CHAR_COLL_NAME,
DTD2.DTD_IDENTIFIER AS XML_CHAR_DTD_ID,
P.XML_OPTION, P.XML_PASSING_MECHANISM
```

Augment the LEFT JOIN with the following additional LEFT JOIN

```
LEFT JOIN
  DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR DTD2
  ON ( P.SPECIFIC_CATALOG, P.SPECIFIC_SCHEMA, P.SPECIFIC_NAME,
       'ROUTINE', P.XML_STRING_DTD_IDENTIFIER )
   = ( DTD2.OBJECT_CATALOG, DTD2.OBJECT_SCHEMA, DTD2.OBJECT_NAME,
       DTD2.OBJECT_TYPE, DTD2.DTD_IDENTIFIER )
```

Conformance Rules

1) Insert this CR Without Feature X160, "Basic Information Schema for registered XML Schemas", conforming SQL language shall not reference reference INFORMATION_SCHEMA.PARAMETERS.XML_SCHEMA_CATALOG, INFORMATION_SCHEMA.PARAMETERS.XML_SCHEMA_SCHEMA, or INFORMATION_SCHEMA.PARAMETERS.XML_SCHEMA_NAME.

2) Insert this CR Without Feature X161, "Advanced Information Schema for registered XML Schemas", conforming SQL language shall not reference reference INFORMATION_SCHEMA.PARAMETERS.XML_SCHEMA_NAMESPACE or INFORMATION_SCHEMA.PARAMETERS.XML_SCHEMA_ELEMENT.

21.11 ROUTINES view

This Subclause modifies Subclause 5.54, "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

Add the following columns to the <select list> following D.DTD_IDENTIFIER

```
        ,
        D.XML_PRIMARY_MODIFIER, D.XML_SECONDARY_MODIFIER,
        D.XML_SCHEMA_CATALOG, D.XML_SCHEMA_SCHEMA, D.XML_SCHEMA_NAME,
        D.XML_SCHEMA_NAMESPACE, D.XML_SCHEMA_ELEMENT,
```

Add the following columns to the <select list> following DT.DTD_IDENTIFIER AS RESULT_CAST_DTD_IDENTIFIER

```
        ,
        DT.XML_PRIMARY_MODIFIER AS RESULT_CAST_XML_PRIMARY_MODIFIER,
        DT.XML_SECONDARY_MODIFIER AS RESULT_CAST_XML_SECONDARY_MODIFIER,
        DT.XML_SCHEMA_CATALOG AS RESULT_CAST_XML_SCHEMA_CATALOG,
        DT.XML_SCHEMA_SCHEMA AS RESULT_CAST_XML_SCHEMA_SCHEMA,
        DT.XML_SCHEMA_NAME AS RESULT_CAST_XML_SCHEMA_NAME,
        DT.XML_SCHEMA_NAMESPACE AS RESULT_CAST_XML_SCHEMA_NAMESPACE,
        DT.XML_SCHEMA_ELEMENT AS RESULT_CAST_XML_SCHEMA_ELEMENT,
        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
```

Replace the first line of the <from clause> with the following

```
    FROM ( ( DEFINITION_SCHEMA.ROUTINES AS R
```

Append the following JOIN to the <from clause>

```
      LEFT JOIN
           DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS 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 ) )
```

NOTE 110 — The ROUTINES view contains three sets of columns that each describe a data type. The set of columns that are prefixed with "XML_" describes 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

1) Insert this CR Without Feature X160, “Basic Information Schema for registered XML Schemas”, conforming SQL language shall not reference INFORMATION_SCHEMA.ROUTINES.XML_SCHEMA_CATALOG, INFORMATION_SCHEMA.ROUTINES.XML_SCHEMA_SCHEMA, INFORMATION_SCHEMA.ROUTINES.XML_SCHEMA_NAME, NFORMATION_SCHEMA.ROUTINES.RESULT_CAST_XML_SCHEMA_CATALOG, INFORMATION_SCHEMA.ROUTINES.RESULT_CAST_XML_SCHEMA_SCHEMA, or INFORMATION_SCHEMA.ROUTINES.RESULT_CAST_XML_SCHEMA_NAME.

2) Insert this CR Without Feature X161, “Advanced Information Schema for registered XML Schemas”, conforming SQL language shall not reference INFORMATION_SCHEMA.ROUTINES.XML_SCHEMA_NAMESPACE, INFORMATION_SCHEMA.ROUTINES.XML_SCHEMA_ELEMENT, INFORMATION_SCHEMA.ROUTINES.RESULT_CAST_XML_SCHEMA_NAMESPACE, or INFORMATION_SCHEMA.ROUTINES.RESULT_CAST_XML_SCHEMA_ELEMENT.

21.12 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_SCHEMA_ELEMENTS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature X161, “Advanced Information Schema for registered XML Schemas”, conforming SQL language 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.

21.13 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,
           XSN. XML_SCHEMA_NAMESPACE_IS_DETERMINISTIC
    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_SCHEMA_NAMESPACES
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature X161, “Advanced Information Schema for registered XML Schemas”, conforming SQL language 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.

21.14 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, “Basic Information Schema for registered XML Schemas”, conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCHEMAS.

2) Without Feature X161, “Advanced Information Schema for registered XML Schemas”, conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCHEMAS.SCHEMA_IS_DETERMINISTIC.

3) Without Feature F391, “Long identifiers”, conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCHEMAS.

21.15 Short name views

This Subclause modifies Subclause 5.82, "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

Append the following to the <view column list> of ATTRIBUTES_S view

```
,
XML_PRIMARY_MOD,     XML_SECONDARY_MOD,  XML_SCHEMA_CATALOG,
XML_SCHEMA_SCHEMA,   XML_SCHEMA_NAME,    XML_SCHEMA_NSPACE,
XML_SCHEMA_ELEMENT
```

Append the following to the <select list> of ATTRIBUTES_S view definition

```
,
XML_PRIMARY_MODIFIER, XML_SECONDARY_MODIFIER, XML_SCHEMA_CATALOG,
XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE,
XML_SCHEMA_ELEMENT
```

Append the following to the <view column list> of COLUMN_S view

```
,
XML_PRIMARY_MOD,     XML_SECONDARY_MOD,  XML_SCHEMA_CATALOG,
XML_SCHEMA_SCHEMA,   XML_SCHEMA_NAME,    XML_SCHEMA_NSPACE,
XML_SCHEMA_ELEMENT
```

Append the following to the <select list> of COLUMN_S view definition

```
,
XML_PRIMARY_MODIFIER, XML_SECONDARY_MODIFIER, XML_SCHEMA_CATALOG,
XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE,
XML_SCHEMA_ELEMENT
```

Append the following to the <view column list> of DOMAIN_S view

```
,
XML_PRIMARY_MOD,     XML_SECONDARY_MOD,  XML_SCHEMA_CATALOG,
XML_SCHEMA_SCHEMA,   XML_SCHEMA_NAME,    XML_SCHEMA_NSPACE,
XML_SCHEMA_ELEMENT
```

Append the following to the <select list> of DOMAIN_S view definition

```
,
XML_PRIMARY_MODIFIER, XML_SECONDARY_MODIFIER, XML_SCHEMA_CATALOG,
XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE,
XML_SCHEMA_ELEMENT
```

Append the following to the <view column list> of ELEMENT_TYPES_S view

```
,
XML_PRIMARY_MOD,     XML_SECONDARY_MOD,  XML_SCHEMA_CATALOG,
```

```
        XML_SCHEMA_SCHEMA,  XML_SCHEMA_NAME,    XML_SCHEMA_NSPACE,
        XML_SCHEMA_ELEMENT
```

Append the following to the <select list> of ELEMENT_TYPES_S view definition

```
        ,
        XML_PRIMARY_MODIFIER, XML_SECONDARY_MODIFIER, XML_SCHEMA_CATALOG,
        XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE,
        XML_SCHEMA_ELEMENT
```

Append the following to the <view column list> of FIELDS_S view

```
        ,
        XML_PRIMARY_MOD,    XML_SECONDARY_MOD,  XML_SCHEMA_CATALOG,
        XML_SCHEMA_SCHEMA,  XML_SCHEMA_NAME,    XML_SCHEMA_NSPACE,
        XML_SCHEMA_ELEMENT
```

Append the following to the <select list> of FIELDS_S view definition

```
        ,
        XML_PRIMARY_MODIFIER, XML_SECONDARY_MODIFIER, XML_SCHEMA_CATALOG,
        XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE,
        XML_SCHEMA_ELEMENT
```

Append the following to the <view column list> of METHOD_SPEC_PARAMS_S view

```
        ,
        XML_PRIMARY_MOD,    XML_SECONDARY_MOD,  XML_SCHEMA_CATALOG,
        XML_SCHEMA_SCHEMA,  XML_SCHEMA_NAME,    XML_SCHEMA_NSPACE,
        XML_SCHEMA_ELEMENT
```

Append the following to the <select list> of METHOD_SPEC_PARAMS_S view definition

```
        ,
        XML_PRIMARY_MODIFIER, XML_SECONDARY_MODIFIER, XML_SCHEMA_CATALOG,
        XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE,
        XML_SCHEMA_ELEMENT
```

Append the following to the <view column list> of METHOD_SPECS_S view

```
        ,
        XML_PRIMARY_MOD,    XML_SECONDARY_MOD,  XML_SCHEMA_CATALOG,
        XML_SCHEMA_SCHEMA,  XML_SCHEMA_NAME,    XML_SCHEMA_NSPACE,
        XML_SCHEMA_ELEMENT, RC_XML_PRIMARY_MOD, RC_XML_SECOND_MOD,
        RC_XML_SCHEMA_CAT,  RC_XML_SCH_SCH,     RC_XML_SCH_NAME,
        RC_XML_SCH_NSPACE,  RC_XML_SCH_ELEMENT
```

Append the following to the <select list> of METHOD_SPECS_S view definition

```
        ,
        XML_PRIMARY_MODIFIER, XML_SECONDARY_MODIFIER, XML_SCHEMA_CATALOG,
        XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE,
        XML_SCHEMA_ELEMENT, RESULT_CAST_XML_PRIMARY_MODIFIER,
        RESULT_CAST_XML_SECONDARY_MODIFIER,
        RESULT_CAST_XML_SCHEMA_CATALOG, RESULT_CAST_XML_SCHEMA_SCHEMA,
        RESULT_CAST_XML_SCHEMA_NAME, RESULT_CAST_XML_SCHEMA_NAMESPACE,
        RESULT_CAST_XML_SCHEMA_ELEMENT
```

Append the following to the <view column list> of PARAMETERS_S view

```
        ,     XML_PRIMARY_MOD,    XML_SECONDARY_MOD,
```

```
        XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA,  XML_SCHEMA_NAME,
        XML_SCHEMA_NSPACE,  XML_SCHEMA_ELEMENT, 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
```

Append the following to the <select list> of PARAMETERS_S view definition

```
        , XML_PRIMARY_MODIFIER, XML_SECONDARY_MODIFIER,
        XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME,
        XML_SCHEMA_NAMESPACE, XML_SCHEMA_ELEMENT, 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
```

Add the following columns to the <view column list> of the ROUTINES_S view, following DTD_IDENTIFIER

```
        XML_PRIMARY_MOD,    XML_SECOND_MOD,
        XML_SCHEMA_CAT,     XML_SCH_SCH,        XML_SCH_NAME,
        XML_SCH_NSPACE,     XML_SCH_ELEMENT,
```

Add the following columns to the <view column list> of the ROUTINES_S view, following RC_DTD_IDENTIFIER

```
        ,
        RC_XML_PRIMARY_MOD, RC_XML_SECOND_MOD,  RC_XML_SCHEMA_CAT,
        RC_XML_SCH_SCH,     RC_XML_SCH_NAME,    RC_XML_SCH_NSPACE,
        RC_XML_SCH_ELEMENT, 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_MECH
```

Add the following to the <select list> of ROUTINES_S view definition following DTD_IDENTIFIER

```
        XML_PRIMARY_MODIFIER, XML_SECONDARY_MODIFIER,
        XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME,
        XML_SCHEMA_NAMESPACE, XML_SCHEMA_ELEMENT,
```

Add the following to the <select list> of ROUTINES_S view definition following RESULT_CAST_DTD_IDENTIFIER

```
        ,
        RESULT_CAST_XML_PRIMARY_MODIFIER,
        RESULT_CAST_XML_SECONDARY_MODIFIER,
        RESULT_CAST_XML_SCHEMA_CATALOG, RESULT_CAST_XML_SCHEMA_SCHEMA,
        RESULT_CAST_XML_SCHEMA_NAME, RESULT_CAST_XML_SCHEMA_NAMESPACE,
        RESULT_CAST_XML_SCHEMA_ELEMENT, 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
```

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,
           XML_SCH_NSP_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_NAMESPACE_IS_DETERMINISTIC
    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, “Basic Information Schema for registered XML Schemas”, conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCHEMAS_S.

2) Insert this CR Without Feature X161, “Advanced Information Schema for registered XML Schemas”, conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCH_ELEMENTS_S.

3) Insert this CR Without Feature X161, “Advanced Information Schema for registered XML Schemas”, conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCH_NSPACES_S.

4) Insert this CR Without Feature X161, “Advanced Information Schema for registered XML Schemas”, conforming SQL language shall not reference INFORMATION_SCHEMA.XML_SCHEMAS_S.XML_SCHEMA_IS_DET.

(Blank page)

22 Definition Schema

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

22.1 DATA_TYPE_DESCRIPTOR base table

This Subclause modifies Subclause 6.22, "DATA_TYPE_DESCRIPTOR base table", in ISO/IEC 9075-11.

Function

The DATA_TYPE_DESCRIPTOR table has one row for each usage of a datatype as identified by ISO/IEC 9075. It effectively contains a representation of the data type descriptors.

Definition

Add the following column definitions

```
XML_PRIMARY_MODIFIER        INFORMATION_SCHEMA.CHARACTER_DATA,
XML_SECONDARY_MODIFIER      INFORMATION_SCHEMA.CHARACTER_DATA,
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,
```

Augment constraint DATA_TYPE_DESCRIPTOR_DATA_TYPE_CHECK_COMBINATIONS:

Add the following predicate to each OR clause excepting the final OR clause of the constraint:

```
AND
  ( XML_PRIMARY_MODIFIER, XML_SECONDARY_MODIFIER, XML_SCHEMA_CATALOG,
    XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE,
    XML_SCHEMA_ELEMENT ) IS NULL
```

Add the following OR clause to the end of the constraint:

```
OR
  ( DATA_TYPE = 'XML'
  AND
    ( XML_PRIMARY_MODIFIER IN ( 'DOCUMENT', 'CONTENT', 'SEQUENCE' ) )
    AND
      ( XML_SECONDARY_MODIFIER IN ( 'UNTYPED', 'ANY', 'XMLSCHEMA' ) )
  AND
    XML_PRIMARY_MODIFIER IS NOT NULL
  AND
    ( XML_PRIMARY_MODIFIER NOT IN ( 'DOCUMENT', 'CONTENT' )
```

```
        OR
          XML_SECONDARY_MODIFIER IS NOT NULL )
      AND
          ( XML_SECONDARY_MODIFIER <> 'XMLSCHEMA'
          OR
            ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME ) IS NULL )
      AND
        ( CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
          CHARACTER_OCTET_LENGTH, CHARACTER_MAXIMUM_LENGTH,
          COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
            IS NULL
      AND
        ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE,
          DECLARED_DATA_TYPE, DECLARED_NUMERIC_PRECISION,
          DECLARED_NUMERIC_SCALE ) IS NULL
      AND
        DATETIME_PRECISION IS NULL
      AND
        ( 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 )
```

Add 'XML' to the IN list of the final OR clause of the constraint

Add the following check constraint at the end of the constraints:

```
,
CONSTRAINT
    DATA_TYPE_DESCRIPTOR_CHECK_REFERENCES_XML_SCHEMA_NAMESPACES
  CHECK ( XML_SCHEMA_CATALOG NOT IN
          ( SELECT CATALOG_NAME
            FROM SCHEMATA )
        OR
          ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME,
            XML_SCHEMA_NAMESPACE) IN
            ( SELECT XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA,
                     XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE
              FROM XML_SCHEMA_NAMESPACES )
          )
```

Add the following check constraint at the end of the constraints:

```
,
CONSTRAINT
    DATA_TYPE_DESCRIPTOR_CHECK_REFERENCES_XML_SCHEMA_ELEMENTS
  CHECK ( XML_SCHEMA_CATALOG NOT IN
          ( SELECT CATALOG_NAME
            FROM SCHEMATA )
        OR
          ( XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME,
            XML_SCHEMA_NAMESPACE, XML_SCHEMA_ELEMENT) IN
            ( SELECT XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA,
```



```
            XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE,
            XML_SCHEMA_ELEMENT
      FROM XML_SCHEMA_ELEMENTS )
)
```

Description

1) Insert this Descr. Case:

   a) If DATA_TYPE is 'XML', then:

      i) The value of XML_PRIMARY_MODIFIER is 'DOCUMENT', 'CONTENT', or 'SEQUENCE'.

      ii) If the value of XML_PRIMARY_MODIFIER is either 'DOCUMENT' or 'CONTENT', then the value of XML_SECONDARY_MODIFIER is 'UNTYPED', 'ANY', or 'XMLSCHEMA'; otherwise, the value of XML_SECONDARY_MODIFIER is the null value.

      iii) Case:

         1) If the value of XML_SECONDARY_MODIFIER is 'XMLSCHEMA', then

            A) The values of XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, and XML_SCHEMA_NAME are the catalog name, the unqualified schema name, and the name of a registered XML Schema, respectively, identified by the registered XML Schema descriptor included in the XML type descriptor.

            B) The value of XML_SCHEMA_NAMESPACE is:

               Case:

               I) If the XML type descriptor contains an XML namespace URI *NS*, then *NS*.

               II) Otherwise, the null value.

            C) The value of XML_SCHEMA_ELEMENT is:

               Case:

               I) If the XML type descriptor contains an XML NCName *EN*, then *EN*.

               II) Otherwise, the null value.

         2) Otherwise, the values of XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE, and XML_SCHEMA_ELEMENT are the null value.

   b) Otherwise, the values of XML_PRIMARY_MODIFIER, XML_SECONDARY_MODIFIER, XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME, XML_SCHEMA_NAMESPACE and XML_SCHEMA_ELEMENT are the null value.

22.2 PARAMETERS base table

This Subclause modifies Subclause 6.33, "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

Append the following <column definition>s

```
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' ) ),
```

Append the following <table constraint definition>

```
,
CONSTRAINT PARAMETERS_CHECK_XML_DATA_TYPE
  CHECK (
    ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA,
      SPECIFIC_NAME, 'ROUTINE', XML_STRING_DTD_IDENTIFIER ) IN
      ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA,
               OBJECT_NAME, OBJECT_TYPE, DTD_IDENTIFIER
        FROM DATA_TYPE_DESCRIPTOR ) )
```

Description

1) Insert this Descr. 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.

   NOTE 111 — The meaning of the term "DTD" used in "XML_STRING_DTD_IDENTIFIER" and "DTD_IDENTIFIER" is different from the meaning of the term "DTD" defined in Subclause 3.1.1, "Definitions taken from XML".

2) Insert this Descr. The value of XML_OPTION is the associated XML option of the parameter being described.

3) Insert this Descr. 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.

22.3 ROUTINES base table

This Subclause modifies Subclause 6.45, "ROUTINES base table", in ISO/IEC 9075-11.

Function

The ROUTINES base table has one row for each SQL-invoked routine.

Definition

Append the following <column definition>s

```
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' ) ),
```

Append the following <table constraint definition>

```
,
CONSTRAINT ROUTINES_CHECK_XML_STRING_DTD_IDENTIFIER
  CHECK (
    ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
      'ROUTINE', XML_STRING_DTD_IDENTIFIER ) IN
      ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
               OBJECT_TYPE, DTD_IDENTIFIER
        FROM DATA_TYPE_DESCRIPTOR ) )
```

Description

1) Insert this Descr. 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.

   NOTE 112 — The meaning of the term "DTD" used in "XML_STRING_DTD_IDENTIFIER" and "DTD_IDENTIFIER" is different from the meaning of the term "DTD" defined in Subclause 3.1.1, "Definitions taken from XML".

2) Insert this Descr. The value of XML_OPTION is the associated XML option of the result of SQL-invoked routine being described.

3) Insert this Descr. 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.

22.4 SQL_CONFORMANCE base table

This Subclause modifies Subclause 6.48, "SQL_CONFORMANCE base table", in ISO/IEC 9075-11.

Function

The SQL_CONFORMANCE base table has one row for each conformance element (part, feature, and subfeature) identified by ISO/IEC 9075.

Definition

```
No additional Definition.
```

Description

No additional Descriptions.

Table Population

1) Insert this specification There is one row in this table for every row in Table 23, "Feature taxonomy for optional features", in [ISO9075-14]. In each such row:

   a) TYPE is 'FEATURE'.

   b) ID is the value of "Feature ID" from Table 23, "Feature taxonomy for optional features", in [ISO9075-14].

   c) NAME is the value of "Feature Name" from Table 23, "Feature taxonomy for optional features", in [ISO9075-14].

   d) SUB_ID and SUB_NAME are each a character string consisting of a single space.

   e) IS_SUPPORTED, IS_VERIFIED_BY, and COMMENTS are as described by Description 4), Description 5), and Description 7).

22.5 USAGE_PRIVILEGES base table

This Subclause modifies Subclause 6.64, "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

Augment the <in value list> in constraint USAGE_PRIVILEGES_OBJECT_TYPE_CHECK

```
, 'XML SCHEMA'
```

Augment the <table subquery> in constraint USAGE_PRIVILEGES_CHECK_REFERENCES_OBJECT

```
UNION
  SELECT XML_SCHEMA_CATALOG, XML_SCHEMA_SCHEMA, XML_SCHEMA_NAME,
         'XML SCHEMA'
  FROM XML_SCHEMAS
```

Description

1) Augment Desc. 4).

XML SCHEMA	The object to which the privilege applies is a registered XML Schema.

22.6 XML_SCHEMA_ELEMENTS base table

Function

The XML_SCHEMA_ELEMENTS base table has one row for each global element declaration schema 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_ELEMENT_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.

NOTE 113 — The concept of a global element declaration schema component being non-deterministic is defined in Subclause 4.2.6, "Registered XML Schemas".