Title: XMLQuery

Author: Fred Zemke, for the H2 ad hoc subcommittee on SQL/XML
Source: U.S.A.
Status: Change proposal
Date: March 14, 2004

Abstract

XQuery is a query language for XML under development by W3C. This paper, the culmination of a series of three papers, proposes a new pseudofunction, XMLQuery, which will evaluate an XQuery expression within SQL.

changes in r1: change ANYTYPE to ANY; editorial improvements

References

[Foundation:2003] Jim Melton (ed), "ISO International Standard (IS) Database Language SQL - Part 2: SQL/Foundation", ISO/IEC 9075-2:2003

[Schemata:2003] Jim Melton (ed), "ISO International Standard (IS) Database Language SQL - Part 11: Information and Definition Schemas (SQL/Schemata)", ISO/IEC 9075-2:2003

[SQL/XML:2003] Jim Melton (ed), "ISO International Standard (IS) Database Language SQL - Part 14: SQL/XML", ISO/IEC 9075-14:2003

[Foundation WD] Jim Melton (ed), "Working Draft (WD) Database Language SQL - Part 2: SQL/Foundation", ISO/IEC JTC1/SC32 WG3:HBA-003 = ANSI INCITS H2-2003-305

[SQL/XML WD] Jim Melton (ed.), "Working Draft (WD) XML-Related Specifications (SQL/XML)", ISO/IEC JTC1/SC32 WG3:SIA-010 = ANSI INCITS H2-2003-427

[SQL:2003 TC] Stephen Cannan (ed.), "Draft Technical Corrigendum", ISO/IEC JTC1/SC32 WG3:HBA-011

[XML 1.0] "Extensible Markup Language (XML) 1.0 (third edition)", W3C Recommendation, 4 Octoboer 2004, available at http://www.w3.org/TR/2000/REC-xml-20040204

[XML 1.1] "Extensible Markup Language (XML) 1.1", W3C Recommendation, 15 Octoboer 2002, available at http://www.w3.org/TR/2002/CR-xml11-20040204

[Namespaces 1.0] "Namespaces in XML", W3C Recommendation, 14 January 1999, available at http://www.w3.org/TR/1999/REC-xml-names-19990114

[Namespaces 1.1] "Namespaces in XML 1.1", W3C Candidate Recommendation, 18 December 2002, available at http://www.w3.org/TR/2002/CR-xml-names11-20021218

[XQuery Language] "XQuery 1.0: an XML query language", W3C working draft, 12 November 2003, available at http://www.w3.org/TR/xquery/

[XQuery DM] "XQuery 1.0 and XPath 2.0 Data Model", W3C working draft, 12 November 2003, available at http://www.w3.org/TR/xpath-datamodel/

[XQuery F&O] "XQuery 1.0 and XPath 2.0 Functions and Operators", W3C working draft, 12 November 2003, available at http://www.w3.org/TR/xpath-functions/

[Serialization] "XSLT 2.0 and XQuery 1.0 Serializtion", W3C working draft, 12 November 2003, available at http://www.w3c.org/TR/xslt-xquery-serialization

[H2-2004-019] Fred Zemke, "Moving to the XQuery data model", ANSI INCITS H2-2004-019

[H2-2004-020] Fred Zemke, "XMLCast", ANSI INCITS H2-2004-020

1. XMLQuery

This is the third in a series of three papers. It assumes [H2-2004-019] and [H2-2004-020] as prerequisites.

XQuery is a query language for XML being developed by the World Wide Web Consortium (W3C). XQuery is defined by a suite of specifications: a data model [XQuery DM], a language specification [XQuery], functions and operators [XQuery F&O], formal semantics [XQuery FS], and serialization [Serialization].

This paper defines a pseudofunction, XMLQuery, to invoke XQuery from SQL. Together with XMLCast, specified in [H2-2004-020], these pseudofunctions will enable the user to query XML data stored in a relational database. They will also serve as a platform for defining XMLTable, an operator to construct a relational view of XML data, in a subsequent paper.

1.1 XQuery and SQL, a comparison

XQuery is an "expression" language rather than a "statement" language. That is, the fundamental unit of computation is the expression. Expressions may be arbitrarily nested, and there is a precedence hierarchy encompassing all varieties of expressions. Currently, XQuery is a read-only language, with no update capability.

In addition, XQuery is a functional language, which means that the value of a site never changes during its lifetime. For example, the XQuery "let" statement, which looks like an assignment

statement, does not actually mutate the value of its left-hand side; instead, it creates a nested context in which a new variable by that name is assigned a value. Ironically, the value of a variable never varies.

[XQuery] section 2.2 “Processing model” has an interesting diagram and description of the overall model for evaluating an XQuery expression. There are two phases, the static analysis phase (analogous to the Syntax Rules in SQL) and the dynamic evaluation phase (analogous to the General Rules in SQL). Unlike SQL, it is an implementation-defined choice, whether to perform type checking during static analysis or during evaluation.

XQuery has no formal analog to our Access Rules, though implementations could certainly provide access checking at the points where metadata and data crosses the boundary from the external environment to the XQuery environment (for example, in populating data instances and importing schemas and function libraries).

Analogous to SQL’s Conformance Rules, XQuery has four optional features, called the Schema Import, Static Typing, Full Axis and Module Features.

The specification of XQuery is split over five volumes: the data model [XQuery DM], the language [XQuery], functions and operators [XQuery F&O], serialization [Serialization] and formal semantics [XQuery FS]. These are not analogous to the parts of SQL, because all of them must be read in conjunction with one another. Thus the union of these five parts is roughly analogous to the first 10 Clauses of SQL/Foundation (and roughly the same the bulk when printed). Basically:

— [XQuery DM] is somewhat analogous to Subclause 4.1 “Data types” in Foundation.

— [Serialization] might be used as bindings for output (though one could also use implementation-defined means for this, as we have done);

— [XQuery] provides the syntax, with some discussion and examples,

— [XQuery FS] attempts to treat everything in [XQuery] with matehmatical rigor by converting XQuery expressions into a core language and heavy use of built-in functions;

— and [XQuery F&O] provides descriptions of all the functions that [XQuery FS] relies on, plus a basic function library exposed to the user.

[XQuery] and [XQuery FS] have a lot of duplication. In general, [XQuery] tries to be colloquial, whereas [XQuery FS] tries to be more precise by being very formal. Since [XQuery FS] is the precise specification of XQuery’s semantics, our proposal primarily references that document.

1.2 Examples

```
XMLQuery ( '//book/author[$n]'
           PASSING BY REF T.Library,
                          S.A + S.B AS "n"
           RETURNING SEQUENCE )
```

This example evaluates the XPath expression /book/author[$n]. The context item for the XPath expression is T.Library, indicated by the fact that this parameter in the PASSING clause has no alias. The other expression in the PASSING clause, S.A + S.B, is computed and the result is used to initialize an XQuery variable, $n, indicated by the alias "n" for the expression in the PASSING clause. The phrase BY REF immediately following the PASSING keyword establishes the default passing mechanism for this invocation, which is BY REF, ie, without making a copy. This passing mechanism becomes the default for any argument of XML type and for the return type. In this example, the default passing mechanism is applicable to T.Library (presumably an XML value) but not to S.A + S.B, which is clearly a numeric value.

```
XMLQuery ( '//book/author[$n]'
           PASSING BY VALUE T.Library,
                            S.A + S.B AS "n"
           RETURNING SEQUENCE BY REF )
```

This example is almost identical to the preceding, except that the default passing mechanism has been changed to BY VALUE. Since a different passing mechanism is desired for the return value, the default passing mechanism is explicitly overridden following RETURNING SEQUENCE.

```
XMLQuery ( '//book/author[$n]'
           PASSING BY VALUE T.Library BY REF,
                            S.A + S.B AS "n"
           RETURNING SEQUENCE )
```

The preceding example shows how to override the default passing mechanism for a particular argument in the PASSING clause.

This example also illustrates returning by value, meaning that a copy is made. When making this copy, for each node in the sequence, the parent property is set to empty. For example, suppose that the value of T.Library is

```
<library>
  <branch name='northside'>
    <book title='Feuerversicherung'>
      <author name='Ludwig von Beethoven/>
    </book>
    <book title='Seguros'>
       <author name='Garcia Ortega y Gasset'/>
    </book>
  </branch>
  <branch name='southside'>
    <book title='Vergangenheit'>
      <author name='Friedrich Nietsche'/>
    </book>
    <book title='Excelsior'>
      <author name='Gabriel Daniel Fahrenheit'/>
    </book>
```

```
  </branch>
</library>
```

If S.A + S.B equals 1, then the XPath expression will select every <author> element in T.Library, for a sequence of four XQuery element nodes. Note that all four nodes are contained in the same XQuery tree, whose root is the document node above the <library> element. If this sequence is returned by reference, that sequence is the return value of the XMLQuery invocation. However, if the return mechanism is BY VALUE, then the result is a sequence of four element nodes with no parent. The difference can be seen by feeding the result back as input to another XMLQuery invocation.

```
XMLQuery ( './parent::node()'
           PASSING BY REF
           XMLQuery ( '//book/author[$n]'
                      PASSING BY REF T.Library,
                                     S.A + S.B AS "n"
                      RETURNING SEQUENCE )
           RETURNING SEQUENCE )
```

The preceding example, all XML values are passed by reference. Consequently the parent axis is preserved from the inner XMLQuery to the outer. With the sample data, the result will be a sequence of four <book> elements, all contained in the original T.Library value.

```
XMLQuery ( './parent::node()'
           PASSING BY VALUE
           XMLQuery ( '//book/author[$n]'
                      PASSING BY REF T.Library,
                                     S.A + S.B AS "n"
                      RETURNING SEQUENCE )
           RETURNING SEQUENCE )
```

In this example, the outer XMLQuery receives by value, and consequently the parent axis of each node received from the inner XMLQuery is empty. Consequently the result of this is the empty XQuery sequence.

```
XMLQuery ( './parent::node()'
           PASSING BY REF
           XMLQuery ( '//book/author[$n]'
                      PASSING BY VALUE T.Library,
                                       S.A + S.B AS "n"
                      RETURNING SEQUENCE )
           RETURNING SEQUENCE )
```

In this example, the inner XMLQuery has passed its value out by value, losing the parent axis on each node in the output sequence. Even though the outer XMLQuery receives by reference, the parent axes have already been lost and the final result is the empty XQuery sequence.

You can also specify RETURNING CONTENT. Since in general the result of an XQuery expression is not a value of XML(CONTENT), the result is placed in an XQuery document constructor, which necessarily creates a copy. Consequently it is forbidden to specify the return passing mechanism with RETURNING CONTENT.

If the PASSING clause is present, then it is mandatory to specify the default passing mechanism. If there are no inputs, then the PASSING clause is omitted, and when RETURNING SEQUENCE, you must specify the return mechanism.

1.3 Syntax

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

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

<XML query parameters> ::=
     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 returning clause> ::=
     RETURNING { CONTENT | SEQUENCE }
```

1.4 Argument passing

An XQuery variable is created for each argument in the PASSING clause that is assigned an <identifier> as a name. For example

```
PASSING T.A + T.B as "sum"
```

creates an XQuery variable called $sum. XQuery variable names are QNames, supporting a namespace prefix. However, the namespace prefix is primarily to support XQuery modules. For simplicity, we only support NCNames ("no-colon" names) in the default namespace as XQuery variable names, enforced through a Syntax Rule.

The XQuery context item does not have a name. To initialize context item, use an expression in the PASSING clause with no name. Naturally, you can have at most one context item.

An important decision was whether node identity is preserved or lost on input to XMLQuery. We decided that reference semantics is useful in some cases (for example, in the forthcoming specification of XMLTable), and value semantics is useful in other cases. We could not agree on a default, which is why there is mandatory syntax to specify the passing mechanism. Believing that the same passing mechanism is probably desired on all inputs, the first syntax after the keyword PASSING must be either BY REF or BY VALUE. Individual arguments may override the default established for the invocation.

1.5 Initializing the XQuery static context

The first step in evaluating an XQuery expression is to initialize the static context. The components of the static context are described in [XQuery] section 2.1.1 "Static context" and appendix C.1, "Static context components", and again in a formal notation in [XQuery FS] 3.1.1 "Static context". Appendix C.1 is the handiest summary, and the only place that describes to what extent an implementation may customize the static context. This appendix has a table, whose first three columns are entitled "Component", "Default predefined value", and "Can be overwritten or augmented by implementation?".

Our initialization of the static context in the Syntax Rules of <XML query> goes through the following stages:

1. First, the components are initialized according to the defaults specified in the second column of the table.
2. Next, the components may be overwritten with implementation-defined values, as permitted by the second column. For example, an implementation may provide built-in namespace prefixes pointing to namespaces containing function libraries or XML Schemas that are provided for the user's convenience.
3. Next, these implementation-defined defaults are further overridden or augmented by our Syntax Rules, as follows:

   a) In-scope namespaces are defined by examining XMLNamespaces declarations in containing scopes of the SQL query. For example:

```
WITH XMLNamespaces ("foo" AS 'http:www.foo.com/bar')
SELECT XMLQuery ('<foo:bar>{.//candy:bar}</foo:bar>'
                 PASSING T.Chocolate
FROM T
```

In this example, the user declares a namespace prefix foo in the WITH clause. Since the XMLQuery invocation is in the <query expression> containing the XMLNamespaces, this namespace prefix is added to the XQuery static context. This enables it to be used in the element constructor for <foo:bar>.

This example could also have been done using an XQuery prolog, which supports (among other things) namespace declarations. The primary benefit of using XMLNamespaces comes if the same namespace prefix must be used in multiple places in the SQL query.

Note that the user's namespace declarations override the ones that are built-in by [XQuery] or the implementation. Thus if the implementation had defined foo, the user's declaration would override it.

b) XQuery variables are created for each SQL expression in the PASSING clause that has been given a name. Below we will look at the XQuery static type that is assigned to these variables.

c) If there is an expression in the PASSING clause with no name, then a context item is created. (Otherwise, there is no context item).

Every value of XQuery has a static type and a dynamic type (analogous respectively to the declared type and most specific type in SQL). For static analysis, we must set the static type of the variable. [XQuery] presents a type syntax known as SequenceType in its section 2.4.3 "SequenceType syntax". [XQuery FS] has a similar, though more detailed, type system, found in section 2.3 "The [XQuery/XPath] type system", which we use in our rules. The XQuery formal type notation can be determined on the basis of the declared type of the SQL expression. We have placed the rules in a separate subclause, since they must be invoked for both the variables and also for the context item.

1.6 Static analysis of the XQuery expression

After initializing the XQuery static context, we can syntax-check the user's query. (Note that we only permit a character string literal; we have not specified a dynamic XQuery capability.) [XQuery] says that an implementation may use either XML 1.0 or XML 1.1 conventions for Name, NCName and QName. In our specification, the Syntax Rules check against XML 1.1 rules, but a Conformance Rule insists on XML 1.0 rules unless XML 1.1 is supported.

Certain EBNF rules in [XQuery] have been labeled with the notation "/* gn: xml-version*/", namely rule [11] "S", rule [19] "NCName", rule [21] "QName", and rule [26] "Char". This so-called "grammar note" is explained in Appendix A.1 "Grammar notes" as meaning that

> The general rules for [XML 1.1] vs. [XML 1.0], as described in the A.2 Lexical structure section, should be applied to this production.

The cross-referenced section says

> It is implementation defined whether legal characters in an XQuery expression are those characters allowed in [XML 1.0] or the larger set allowed in [XML 1.1].

Actually, the legal characters in [XML 1.0] and [XML 1.1] are the same (this appears to be a mis-statement on the part of XQuery working group). However, to make sense of the grammar note, it appears that what [XQuery] means to say is that it is implementation-defined whether the four nonterminals S, NCName, QName and Char are defined by reference to [XML 1.0] or [XML 1.1].

1.7 Evaluating the XQuery expression

The General Rules of <XML query> must evaluate the XQuery expression. To do this, we must first initialize the dynamic context, which is described in [XQuery] section 2.1.2 “Dynamic context”, appendix C.2 “Dynamic context components”, and [XQuery FS] section 3.1.2 “Dynamic context”. The table in [XQuery] appendix C.2 provides a convenient summary of which components may be overridden or augmented by the implementation. As with the static context, we do this in stages, first with the defaults specified by [XQuery], then with any implementation-defined overrides, and finally with our own overrides.

The latter consists of creating XQuery variables with the right values. The values are obtained from SQL expressions in the PASSING clause by casting to XML(SEQUENCE), as anticipated in the discussion of the casting rules. If the result is null, the null value is passed in as an empty XQuery sequence, since there is no null value in XQuery.

The context item is represented by a special built-in variable called *fs*:dot, in a fictitious namespace (hence the italicized prefix *fs* for the Formal Semantics namespace). The context item may only be an XQuery item; that is, it may not be an empty sequence, and it may not be a sequence of more than one item. We have specified that if the cast to XML(SEQUENCE) results in a null value or the empty sequence, then XQuery is not actually invoked, and the result is the null value. Attempting to pass more than one item as the context item raises an exception.

After creating the dynamic context, we can evaluate the XQuery expression, again using either XML 1.0 or 1.1 lexical rules. (The issue is not entirely resolved during syntax check because XQuery computed element and attribute constructors can construct a QName dynamically, which might raise an error). Since General Rules are not modified by Conformance Rules, the General Rules must test for support for XML 1.1 explicitly.

Evaluating an XQuery expression may result in either a type error or a dynamic error. Unfortunately, there is not currently a good mechanism to pass along the complete identifier of an XQuery error using the SQL diagnostics system. Consequently we have specified a single SQL error to report all XQuery errors.

1.8 Output

We provide three ways to return a value from XMLQuery:

a) RETURNING CONTENT: User cannot specify BY VALUE, which is always implicit. The result is coerced into an XQuery document node using the "normalization" technique found in [Serialization].

b) RETURNING SEQUENCE BY VALUE: makes a type-preserving copy of the return value, including all its type information, but losing node identity.

c) RETURNING SEQUENCE BY REF: retains node identity and does not change any type information. This mechanism is needed by XMLTable when exiting from the row pattern

We could not agree on a default, so we have provided mandatory syntax for the return type.

2. Conformance features

We propose the following new conformance features:

X201, XMLQuery: RETURNING CONTENT option

X202, XMLQuery: RETURNING SEQUENCE option

3. Incidental changes

We noticed that Subclause 11.3 <XML query options>, has BNF nonterminals called <XML query option[s]>. We decided that we preferred that only BNF nonterminals introduced in the new Subclause for <XML query> should begin with "XML query". Accordingly, <XML query option[s]> is being renamed <XML lexically scoped option[s]>.

4. Future possibilities

This paper is not the final chapter in querying XML. The ad hoc telecon has also been working on an XMLTable operator for the FROM clause, which will provide a relational view of XML data.

5. Proposal conventions

This proposal uses the following conventions:

1. SMALLCAPS	denote numbered editorial instructions;
~~strikeout~~	denotes existing text to be deleted;
bold red	denotes new text to be inserted;
plain	denotes existing text to be retained;

[Note:...] brackets enclose italicized notes to the proposal reader; boxes surround "editing tags," which are part of the document (not instructions to the editor) and may be deleted, inserted, modified or retained, depending on the typeface within the box.

6. Proposal for [SQL/XML WD]

6.1 Changes to 4.2.3, Operations involving XML values

1. APPEND THE FOLLOWING PARAGRAPH:

<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.

6.2 Changes to 5.1 <token> and <separator>

1. ADD THE FOLLOWING <RESERVED WORD>:

```
<reserved word> ::=
      . . .
      | XMLQUERY
```

2. ADD THE FOLLOWING <NON-RESERVED WORD>:

```
<non-reserved word> ::=
      . . .
      | PASSING
```

6.3 Changes to 6.7 <XML value function>

1. NOTE TO EDITOR: THE EDITS SHOWN BELOW COMPLETELY SUBSUME THE EDITS OF [H2-2004-019]. THIS PAPER SHOWS THE CORRECT MERGER OF BOTH PAPERS FOR THIS SUBCLAUSE.

2. DELETE <XML ROOT> FROM THE BNF FOR <XML VALUE FUNCTION>:

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

3. EDIT SYNTAX RULE 1) AS FOLLOWS:

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

4. EDIT GENERAL RULE 1) AS FOLLOWS:

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

6.4 New Subclause 6.n <XML query>

1. ADD THE FOLLOWING SUBCLAUSE IN ALPHABETIC ORDER FOLLOWING 6.13, "<XML PARSE>":

6.n <XML query>

Function

Evaluate an XQuery expression.

Format

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

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

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

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

<XML query argument> ::=
       <XML query context item>
     | <XML query variable>

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

<XML query variable> ::=
     <value expression> AS <identifier>
```

```
      [ <XML passing mechanism > ]

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

Syntax Rules

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

2) If <XML query parameters> is specified, let *DPM* be the <XML query default passing mechanism> immediately contained in the <XML query parameters>.

3) Case:

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

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

c) Otherwise, <XML query parameters> shall be specified. Let *XQRM* be *DPM*.

4) An XQuery static context *XSC* is created as follows:

a) *XSC* is initially created by applying the Syntax Rules of Subclause 10.n, "Creation of an XQuery execution content", with the <XML query> as the *BNF*.

b) The in-scope variables component of *XSC* is modified as follows:

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

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

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

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

4) The declared type of *VVE* shall be convertible to XML(SEQUENCE) according to the Syntax Rules of 6.n, "<XML cast specification>".

5) If the declared type of *VVE* is not an XML type, then <XML passing mechanism> shall not be specified.

6) Let *VFT* be the XQuery formal type notation determined by the Syntax Rules of Subclause 10.n, "Determination of an XQuery formal type notation", with *VVE* as the *SOURCE*.

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

ii) If <XML query context item> is specified, then:

1) There shall be exactly one <XML query context item> *XQCI*.

2) Let *CVE* be the <value expression> simply contained in the <XML query context item>.

3) The declared type of *CVE* shall be convertible to XML(SEQUENCE) according to the Syntax Rules of 6.n, "<XML cast specification>".

4) If the declared type of *CVE* is not an XML type, then <XML passing mechanism> shall not be specified.

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

6) Let *CFT* be the XQuery formal type notation determined by the Syntax Rules of Subclause 10.n, "Determination of an XQuery formal type notation", with *VVE* as the *SOURCE*.

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

NOTE nnn: initialization of the XQuery static context can in many cases be overridden by the prolog of the <XQuery expression>.

5) The value of the <XQuery expression> shall be an XQuery expression with XML 1.1 lexical rules that passes the XQuery static analysis phase using the XQuery static context *XSC* without raising an XQuery error. It is implementation-defined which optional features of XQuery are supported.

6) The declared type of <XML query> is

Case:

a) If RETURNING CONTENT is specified, then XML(UNTYPED CONTENT).

b) Otherwise, XML(SEQUENCE).

Access Rules

None

General Rules

1) Let *XDC* be an XQuery dynamic context created by applying the General Rules of Subclause 10.n, "Creation of an XQuery execution context".

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

b) Case:

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

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

Case:

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

2) Otherwise,

A) Case:

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

Case:

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

2) Otherwise, the result of applying the General Rules of Subclause 10.n, "Constructing a copy of an XML value", with the value of *CVE* as the *VALUE*.

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

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

B) Case:

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

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 is raised: *data exception — invalid XQuery context item.*

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

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

Case:

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

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

Case:

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

B) Otherwise, the result of applying the General Rules of Subclause 10.n, "Generation of a copy of an XML value", with *V* as the *VALUE*.

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

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

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

2) Case:

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

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

4) Case:

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

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

— Editor's Note —

The XQuery serialization specification is still evolving. We will need to check whether the reference to the XQuery serialization specification in Subclause 6.n, "<XML query>", is still correct and desired. See Possible Problem nnn.

5) Case:

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

b) Otherwise, let *X3* be the result of applying the General Rules of Subclause 10.n, "Constructing a copy of an XML value", with *X2* as the *VALUE*.

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

Conformance Rules

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

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

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

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

6.5 Changes to 7.1 <query expression>

1. EDIT THE BNF FOR <WITH CLAUSE> AS FOLLOWS:

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

2. EDIT THE SYNTAX RULES AS FOLLOWS:

1) | Insert this SR | A <with clause> shall specify an <XML ~~query option~~ **lexically scoped options**> or a <with list> or both.

2) | Insert this SR | The scope of an <XML namespace declaration> contained in an <XML ~~query~~ **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 ~~query~~ **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 ~~query~~ **lexically scoped** options> and <with list>.

3. EDIT THE CONFORMANCE RULES AS FOLLOWS:

1) | Insert this CR | Without Feature X081, "Query-level XML namespace declarations", in conforming SQL language, <with clause> shall not immediately contain an <XML ~~query~~ **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 ~~query~~ **lexically scoped** options> that contains an <XML binary encoding>.

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

6.6 New subclause 10.n, "Determination of an XQuery formal type notation"

1. ADD THE FOLLOWING NEW SUBCLAUSE TO CLAUSE 10, "ADDITIONAL COMMON RULES"

10.n Determination of an XQuery formal type notation

Function

Determine the XQuery formal type notation of an XQuery variable in the XQuery static context.

Format

None

Syntax Rules

1) Let *S* be the *SOURCE* an invocation of this Subclause.

2) Let *SD* be the declared type of *S*.

3) Case:

a) If *S* is a column reference that references a known not null column, then let *BOUNDED* be the empty string.

b) Otherwise, let *BOUNDED* be "?".

4) Let *XFTN* be the XQuery formal type notation determined as follows.

Case:

a) If *SD* is XML(UNTYPED DOCUMENT), then let *XFTN* be "document { element of type xdt:untypedAny} *BOUNDED*".

b) If *SD* is XML(ANY DOCUMENT), then let *XFTN* be "document { element of type xs:anyType} *BOUNDED*".

c) If *SD* is XML(UNTYPED CONTENT), then let *XFTN* be "document { (element of type xdt:untypedAny | comment | processing-instruction | text) * } *BOUNDED*".

d) If *SD* is XML(ANY CONTENT), then let *XFTN* be "document { (element of type xs:anyType | comment | processing-instruction | text) * } *BOUNDED*".

c) If *SD* is XML(SEQUENCE), then let *XFTN* be "(element of type xs:anyType | attribute of type xs:anySimpleType | text | comment | processing-instruction | xdt:anyAtomicType | document { (element of type xs:anyType | | comment | processing-instruction | text) * }) * "

— Editor's Note —

NOTE: [XQuery FS] formal type notations do not currently permit parentheses. This appears to be a bug, which is being reported to W3C. We will need to review the final version of [XQuery FS] to insure that our XQuery formal type notations conform to that specification. We also need to kick W3C's ass if they don't fix this bug. See Possible Problem XML-nnn

d) If *SD* is an SQL predefined type, then let *XMLT* be the result of applying the General Rules of Subclause 9.15, "Mapping SQL data types to XML Schema data types", with *SD* as the SQL type, *ENC* as the *ENCODING*, and "absent" as the *NULLS*. Let *PT* be the XML Schema primitive type that *XMLT* is derived from.

Case:

1) If *SD* is a year-month interval, then let *XFTN* be "xdt:yearMonthDuration *BOUNDED*".

2) If *SD* is a day-time interval, then let *XFTN* be "xdt:dayTimeDuration *BOUNDED*".

3) If *SD* is an exact numeric type with scale 0, then let *XFTN* be "xs:integer *BOUNDED*".

4) Otherwise, let *XFTN* be "*PT BOUNDED*".

5) *XFTN*, or an implementation-defined XML Schema subtype of *XFTN* that describes *S*, is the result of this Subclause.

NOTE nnn: For example, an implementation may use "empty" as the XQuery formal type notation for XMLCast (EMPTY AS XML(SEQUENCE)).

Acces Rules

Nada

General Rules

Nichts

Conformance Rules

None

6.7 Changes to 11.3, <XML query options>

1. CHANGE THE NAME OF THIS SUBCLAUSE TO <XML LEXICALLY SCOPED OPTIONS> (THIS IS A GLOBAL CHANGE THROUGH THE WHOLE DOCUMENT, WHICH SHOULD BE DONE AUTOMATICALLY BY THE TEXT FORMATTING SYSTEM).

11.3 <XML ~~query~~ **lexically scoped** options>

2. GLOBALLY CHANGE <XML QUERY OPTIONS> TO <XML LEXICALLY SCOPED OPTIONS> AND CHANGE <XML QUERY OPTION> TO <XML LEXICALLY SCOPED OPTION>. THIS PROPOSAL INTENDS TO SHOW ALL SUCH CHANGES EXPLICITLY.

3. ACCORDINGLY, EDIT THE FORMAT AS FOLLOWS:

```
<XML ~~query~~ lexically scoped options> ::=
     <XML ~~query~~ lexically scoped option>
     [ <comma> <XML ~~query~~ lexically scoped option> ]

<XML ~~query~~ lexically scoped option> ::=
       <XML namespace declaration>
     | <XML binary encoding>
```

4. EDIT SYNTAX RULE 1) AS FOLLOWS:

   1) An <XML ~~query~~ **lexically scoped** options> shall contain at most one <XML namespace declaration>, and at most one <XML binary encoding>.

5. EDIT SYNTAX RULE 3 AS FOLLOWS:

   3) Each <XML namespace prefix> shall be an XML **1.1** NCName.

6. ADD THE FOLLOWING CONFORMANCE RULE:

   n) Without Feature X211, "XML 1.1 support", in conforming SQL language, each <XML namespace prefix> shall be an XML 1.0 NCName.

6.8 Changes to 12.1 <column definition>

1. EDIT THE FORMAT AS FOLLOWS:

```
<generation expression> ::=
     <left paren>
     [ WITH <XML ~~query~~ lexically scoped options> ]
     <value expression> <right paren>
```

2. EDIT THE SYNTAX RULES AS FOLLOWS:

   1) | Insert this SR | The scope of each <XML namespace declaration item> contained in the <XML ~~query~~ **lexically scoped** options> is the <generation expression>.

   2) | Insert this SR | The scope of an <XML binary encoding> contained in the <XML ~~query~~ **lexically scoped** options> is the <generation expression>.

3. EDIT THE CONFORMANCE RULES AS FOLLOWS:

   1) | Insert this CR | Without Feature X083, "XML namespace declarations in DDL", in conforming SQL language, a <generation expression> shall not immediately contain an <XML ~~query~~ **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 ~~query~~ **lexically scoped** options> that contains an <XML binary encoding>.

      [NOTE to the proposal reader: editorial fix; this CR is lacking a tag.]

   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 ~~the~~ **an** XML type or a distinct type whose source type is ~~the~~ **an** XML type.

[NOTE to the proposal reader: CR 3) is actually edited in the data model migration paper, shown here just to show the merger.]

6.9 Changes to 12.2 <check constraint definition>

1. EDIT THE FORMAT AS FOLLOWS:

```
<check constraint definition> ::=
     CHECK <left paren>
     [ WITH <XML ~~query~~ lexically scoped options> ]
     <search condition> <right paren>
```

2. EDIT THE SYNTAX RULES AS FOLLOWS:

1) | Insert this SR | The scope of each <XML namespace declaration item> contained in the ~~<XML namespace declaration>~~ **<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>.

[NOTE to the proposal reader: missing SR that should have been added when we added <XML binary encoding>.]

3. EDIT THE CONFORMANCE RULES AS FOLLOWS:

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~~ **an** <XML ~~query~~ **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 ~~query~~ **lexically scoped** options> that contains an <XML binary encoding>.

6.10 Changes to 12.4 <assertion definition>

1. EDIT THE FORMAT AS FOLLOWS:

```
<assertion definition> ::=
     CREATE ASSERTION <constraint name> CHECK <left paren>
     [ WITH <XML ~~query~~ lexically scoped options> ]
     <search condition> <right paren>
```

2. EDIT THE SYNTAX RULES AS FOLLOWS:

    1) | Insert this SR | The scope of each <XML namespace declaration item> contained in the <XML ~~query~~ **lexically scoped** options> is the <assertion definition>.

    2) | Insert this SR | The scope of an <XML binary encoding> contained in the <XML ~~query~~ **lexically scoped** options> is the <assertion definition>.

3. EDIT THE CONFORMANCE RULES AS FOLLOWS:

    1) | Insert this CR | Without Feature X083, "XML namespace declarations in DDL", in conforming SQL language, an <assertion definition> shall not immediately contain an <XML ~~query~~ **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 ~~query~~ **lexically scoped** options> that contains an <XML binary encoding>.

6.11 Changes to 13.1, Calls to an <externally-invoked procedure>

1. HARMONIZE THE ADA HEADER WITH THE CHANGES TO SUBCLAUSE 23.1, SQLSTATE.

6.12 Changes to 14.1 <delete statement: searched>

1. EDIT THE FORMAT AS FOLLOWS:

```
<delete statement: searched> ::=
     DELETE [ WITH <XML ~~query~~ lexically scoped options> ]
     FROM <target table> [ WHERE <search condition> ]
```

2. EDIT THE SYNTAX RULES AS FOLLOWS:

    1) | Insert this SR | The scope of each <XML namespace declaration item> contained in the <XML ~~query~~ **lexically scoped** options> is the <delete statement: searched>.

    2) | Insert this SR | The scope of an <XML binary encoding> contained in the <XML ~~query~~ **lexically scoped** options> is the <delete statement: searched>.

3. EDIT THE CONFORMANCE RULES AS FOLLOWS:

    1) | Insert this CR | Without Feature X082, "XML namespace declarations in DML", in conforming SQL language, a <delete statement: searched> shall not immediately contain an <XML ~~query~~ **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 ~~query~~ **lexically scoped** options> that contains an <XML binary encoding>.

6.13 Changes to 14.2 <insert statement>

1. EDIT THE FORMAT AS FOLLOWS:

```
<insert statement> ::=
     INSERT [ WITH <XML ~~query~~ lexically scoped options> ]
     INTO <insertion target> <insert columns and source>
```

2. EDIT THE SYNTAX RULES AS FOLLOWS:

1) | Insert this SR | The scope of each <XML namespace declaration item> contained in <XML ~~query~~ **lexically scoped** options> is the <insert statement>.

2) | Insert this SR | The scope of an <XML binary encoding> contained in <XML ~~query~~ **lexically scoped** options> is the <insert statement>.

3. EDIT THE CONFORMANCE RULES AS FOLLOWS:

1) | Insert this CR | Without Feature X082, "XML namespace declarations in DM"î, in conforming SQL language, an <insert statement> shall not immediately contain an <XML ~~query~~ **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 ~~query~~ **lexically scoped** options> that contains an <XML binary encoding>.

6.14 Changes to 14.3 <merge statement>

1. EDIT THE FORMAT AS FOLLOWS:

```
<merge statement> ::=
     MERGE [ WITH <XML ~~query~~ lexically scoped options> ]
     INTO <target table> [ [ AS ] <correlation name> ]
     USING <table reference> ON <search condition>
     <merge operation specification>
```

2. EDIT THE SYNTAX RULES AS FOLLOWS:

1) | Insert this SR | The scope of each <XML namespace declaration item> contained in <XML ~~query~~ **lexically scoped** options> is the <merge statement>.

2) | Insert this SR | The scope of an <XML binary encoding> contained in <XML ~~query~~**lexically scoped** options> is the <merge statement>.

3. EDIT THE CONFORMANCE RULES AS FOLLOWS:

    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 ~~query~~ **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 ~~query~~ **lexically scoped** options> that contains an <XML binary encoding>.

6.15 Changes to 14.4 <update statement: positioned>

1. EDIT THE FORMAT AS FOLLOWS:

```
<update statement: positioned> ::=
     UPDATE [ WITH <XML query lexically scoped options> ]
     <target table> SET <set clause list>
     WHERE CURRENT OF <cursor name>
```

2. SYNTAX RULES

    1) | Insert this SR | The scope of each <XML namespace declaration item> contained in <XML ~~query~~ **lexically scoped** options> is the <update statement: positioned>.

    2) | Insert this SR | The scope of an <XML binary encoding> contained in <XML ~~query~~ **lexically scoped** options> is the <update statement: positioned>.

3. CONFORMANCE RULES

    1) | Insert this CR | Without Feature X082, "XML namespace declarations in DML", in conforming SQL language, an <update statement: positioned> shall not immediately contain an <XML ~~query~~ **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 ~~query~~ **lexically scoped** options> that contains an <XML binary encoding>.

6.16 Changes to 14.5 <update statement: searched>

1. EDIT THE FORMAT AS FOLLOWS:

```
<update statement: searched> ::=
     UPDATE [ WITH <XML query lexically scoped options> ]
     <target table>
     SET <set clause list> [ WHERE <search condition> ]
```

2. EDIT THE SYNTAX RULES AS FOLLOWS:

   1) | Insert this SR | The scope of each <XML namespace declaration item> contained in <XML ~~query~~ **lexically scoped** options> is the <update statement: searched>.

   2) | Insert this SR | The scope of an <XML binary encoding> contained in <XML ~~query~~ **lexically scoped** options> is the <update statement: searched>.

3. EDIT THE CONFORMANCE RULES AS FOLLOWS:

   1) | Insert this CR | Without Feature X082, "XML namespace declarations in DML", in conforming SQL language, an <update statement: searched> shall not immediately contain an <XML ~~query~~ **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 ~~query~~ **lexically scoped** options> that contains an <XML binary encoding>.

6.17 Changes to 15.1 <compound statement>

1. EDIT THE BNF FOR <COMPOUND STATEMENT> AS FOLLOWS:

```
<compound statement> ::=
     [ <beginning label> <colon> ]
     BEGIN [ [ NOT ] ATOMIC ]
     [ DECLARE <XML ~~query~~ lexically scoped options>
     <semicolon> ]
     [ <local declaration list> ]
     [ <local cursor declaration list> ]
     [ <local handler declaration list> ]
     [ <SQL statement list> ]
     END [ <ending label> ]
```

2. EDIT THE SYNTAX RULES AS FOLLOWS:

   1) | Insert this SR | The scope of each <XML namespace declaration item> contained in <XML ~~query~~ **lexically scoped** options> is the <compound statement>.

   2) | Insert this SR | The scope of an <XML binary encoding> contained in <XML ~~query~~ **lexically scoped** options> is the <compound statement>.

3. EDIT THE CONFORMANCE RULES AS FOLLOWS:

   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 ~~query~~ **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 ~~query~~ **lexically scoped** options> that contains an <XML binary encoding>.

6.18 Changes to 23.1, SQLSTATE

1. ADD THE FOLLOWING EXCEPTIONS TO TABLE 12, "SQLSTATE CLASS AND SUBCLASS VALUES":

Category	Condition	Class	Subcondition	Subclass
X	***data exception***	**22**	***not an XQuery document node***	*editor's choice*
			invalid operand for XML concatenation	*editor's choice*
			not an XQuery document node	*editor's choice*
			invalid XQuery context item	*editor's choice*
			XQuery sequence serialization error	*editor's choice*
W	***warning***	**01**	***XQuery document node was stripped***	*editor's choice*
X	***XQuery error***	*editor's choice*	***(no subclass)***	**000**

6.19 Changes to 24.3 Implied feature relationships of SQL/XML

1. ADD THE INDICATED ROWS TO TABLE 13, "IMPLIED FEATURE RELATIONSHIPS OF SQL/XML":

Feature ID	Feature Name	Implied Feature ID	Implied Feature Name
X201	**XMLQuery: RETURNING CONTENT**	**X010**	**XML type**
X202	**XMLQuery: RETURNING SEQUENCE**	**X190**	**XML(SEQUENCE) type**

6.20 Changes to Annex B, Implementation-defined elements

1. ADD THE FOLLOWING ITEMS:

n.1) Subclause 6.n, "<XML query>"

a) Which optional features of XQuery are supported is implementation-defined.

n.2) Subclause 10.n, "Determination of an XQuery formal type notation"

a) The XQuery formal type notation of an XQuery variable may denote an implementation-defined subtype of the type specified in this Subclause.

6.21 Changes to Annex D, Incompatibilities

1. ADD THE FOLLOWING NEW <RESERVED WORD>S:

XMLQUERY

6.22 Changes to Annex E, SQL/XML feature taxonomy

1. ADD THE INDICATED ROWS TO TABLE 14:

Feature ID	Feature Name
X201	**XMLQuery: RETURNING CONTENT**
X202	**XMLQuery: RETURNING SEQUENCE**

2. ADD THE FOLLOWING POSSIBLE PROBLEMS:

Severity: Possible Problem

Reference: P14, SQL/XML, Subclause 6.n, "<XML query>"

Note at: after GR 4)b)

Source: WG3:SIA-nnn

Possible Problem:

The XQuery serialization specification is still evolving. We will need to check whether the reference to the XQuery serialization specification in Subclause 6.n, "<XML query>", is still correct and desired. See Possible Problem nnn.

Severity: Possible Problem

Reference: P14, SQL/XML, Subclause 10.n, "Determination of an XQuery formal type notation"

Note at: SR 4)c)

Source: WG3:SIA-nnn

Possible Problem:

[XQuery FS] formal type notations do not currently permit parentheses. This appears to be a bug, which is being reported to W3C. We will need to review the final version of [XQuery FS] to insure that our XQuery formal type notations conform to that specification. See Possible Problem XML-nnn

7. Checklist

Concepts	
Access Rules	
Conformance Rules	
Lists of SQL-statements by category	
Table of identifiers used by diagnostics statements	
Collation derivation for character strings	
Closing Possible Problems	
Any new Possible Problems clearly identified	
Reserved and non-reserved keywords	
SQLSTATE tables and Ada package	
Information and Definition Schemas, including short-name views	
Implementation-defined and –dependent Annexes	
Incompatibilities Annex	
Embedded SQL and host language implications	
Dynamic SQL issues: including descriptor areas	
CLI issues	

- End of paper -