XML Query Use Cases

W3C Working Draft 08 June 2001

This version:
http://www.w3.org/TR/2001/WD-xmlquery-use-cases-20010608
Latest version:
http://www.w3.org/TR/xmlquery-use-cases
Previous version:
http://www.w3.org/TR/2001/WD-xmlquery-use-cases-20010215
Editors:
Don Chamberlin (IBM Almaden Research Center) <chamberlin@almaden.ibm.com >
Peter Fankhauser (GMD-IPSI) <fankhaus@darmstadt.gmd.de>
Massimo Marchiori (W3C/MIT/UNIVE) <massimo@w3.org>
Jonathan Robie (Software AG) <jonathan.robie@SoftwareAG-USA.com>

Abstract

This document specifies usage scenarios for the W3C XML Query data model, algebra, and query language.

Status of this document

The material in this document was previously contained in the W3C XML Requirements. Due to its length, we have placed it in a separate document. This is a W3C Working Draft for review by W3C Members and other interested parties. It is a draft document and may be updated, replaced or made obsolete by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". This is work in progress and does not imply endorsement by the W3C membership.

This document has been produced as part of the W3C XML Activity, following the procedures set out for the W3C Process. The document has been written by the XML Query Working Group (W3C members only). The goals of the XML Query working group are discussed in the XML Query Working Group charter (W3C members only).

The XML Query Working Group feels that the contents of this Working Draft are relatively stable, and therefore encourages feedback on this version.

Comments on this document should be sent to the W3C mailing list www-xml-query-comments@w3.org (archived at http://lists.w3.org/Archives/Public/www-xml-query-comments/).

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/.

Table of contents

Appendices

---

1 Use Cases for XML Queries

The use cases listed below were created by the XML Query Working Group to illustrate important applications for an XML query language. Each use case is focused on a specific application area, and contains a Document Type Definition (DTD) and example input data. Each use case specifies a set of queries that might be applied to the input data, and the expected results for each query. Since the English description of each query is concise, the expected results form an important part of the definition of each query, specifying the expected output format. These use cases were originally published as part of the W3C XML Query Requirements document, without solutions in concrete query languages. Now it is being republished with solutions for XQuery.

Several implementors have asked that we make the queries from these use cases available in a separate file to make it easier for them to test their parsers. These queries may be found in [Use Case Sample Queries]. Also, the queries from the XQuery specification itself have been made available in [XQuery Sample Queries].

Some of the use cases assume that input is provided in the form of one or more documents with specific names such as "http://www.bn.com/bib.xml". Other use cases are based on implicit (unnamed) input documents. The input environment for each use case is stated in its Document Type Definition (DTD) section.

These use cases represent a snapshot of an ongoing work. Some important application areas are not yet adequately covered by a use case. The XML Query Working Group reserves the right to add, delete, or modify individual queries or whole use cases as the work progresses. The presence of a query in this set of use cases does not necessarily indicate that the query will be expressible in the XML Query Language(s) to be created by the XML Query Working Group.

1.1 Use Case "XMP": Experiences and Exemplars

This use case contains several example queries that illustrate requirements gathered from the database and document communities.

1.1.1 Document Type Definitions (DTD)

Most of the example queries in this use case are based on a bibliography document named "http://www.bn.com/bib.xml" with the following DTD:

```
<!ELEMENT bib  (book* )>
<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>
<!ATTLIST book  year CDATA  #REQUIRED >
<!ELEMENT author  (last, first )>
<!ELEMENT editor  (last, first, affiliation )>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT last  (#PCDATA )>
<!ELEMENT first  (#PCDATA )>
<!ELEMENT affiliation  (#PCDATA )>
<!ELEMENT publisher  (#PCDATA )>
<!ELEMENT price  (#PCDATA )>
```

1.1.2 Sample Data

Here is the data found at www.bn.com/bib.xml:

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price> 65.95</price>
    </book>

    <book year="1992">
        <title>Advanced Programming in the Unix environment</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>

    <book year="2000">
        <title>Data on the Web</title>
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price> 39.95</price>
    </book>

    <book year="1999">
        <title>The Economics of Technology and Content for Digital TV</title>
        <editor>
               <last>Gerbarg</last><first>Darcy</first>
                <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
    </book>

</bib>
```

1.1.3 DTD for Q5

Q5 also uses information on book reviews and prices from a separate data source named "http://www.amazon.com/reviews.xml" with the following DTD:

```
<!ELEMENT reviews (entry*)>
<!ELEMENT entry   (title, price, review)>
<!ELEMENT title   (#PCDATA)>
<!ELEMENT price   (#PCDATA)>
<!ELEMENT review  (#PCDATA)>
```

1.1.4 Sample Data for Q5

Here are the contents of "http://www.amazon.com/reviews.xml":

```
<reviews>
         <entry>
                  <title>Data on the Web</title>
                  <price>34.95</price>
                  <review>
               A very good discussion of semi-structured database
               systems and XML.
        </review>
         </entry>
         <entry>
                  <title>Advanced Programming in the Unix environment</title>
                  <price>65.95</price>
                  <review>
               A clear and detailed discussion of UNIX programming.
        </review>
         </entry>
         <entry>
                  <title>TCP/IP Illustrated</title>
                  <price>65.95</price>
                  <review>
               One of the best books on TCP/IP.
        </review>
         </entry>
</reviews>
```

1.1.5 DTD for Q9

Q9 uses an input document named "books.xml", with the following DTD:

```
<!ELEMENT chapter (title, section*)>
<!ELEMENT section (title, section*)>
<!ELEMENT title (#PCDATA)>
```

1.1.6 Data for Q9

Here are the contents of books.xml:

```
<chapter>
         <title>Data Model</title>
         <section>
                  <title>Syntax For Data Model</title>
         </section>
         <section>
                  <title>XML</title>
```

```
                    <section>
                             <title>Basic Syntax</title>
                    </section>
                    <section>
                             <title>XML and Semistructured Data</title>
                    </section>
          </section>
</chapter>
```

1.1.7 DTD for Q10

Q10 uses an input document named "prices.xml", with the following DTD:

```
<!ELEMENT prices (book*)>
<!ELEMENT book (title, source, price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT source (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

1.1.8 Data for Q10

Here are the contents of prices.xml:

```
<prices>
          <book>
                    <title>Advanced Programming in the Unix environment</title>
                    <source>www.amazon.com</source>
                    <price>65.95</price>
          </book>
          <book>
                    <title>Advanced Programming in the Unix environment </title>
                    <source>www.bn.com</source>
                    <price>65.95</price>
          </book>
          <book>
                    <title> TCP/IP Illustrated </title>
                    <source>www.amazon.com</source>
                    <price>65.95</price>
          </book>
          <book>
                    <title> TCP/IP Illustrated </title>
                    <source>www.bn.com</source>
                    <price>65.95</price>
          </book>
          <book>
                    <title>Data on the Web</title>
                    <source>www.amazon.com</source>
                    <price>34.95</price>
          </book>
          <book>
                    <title>Data on the Web</title>
                    <source>www.bn.com</source>
                    <price>39.95</price>
          </book>
</prices>
```

1.1.9 Queries and Results

1.1.9.1 Q1

List books published by Addison-Wesley after 1991, including their year and title.

Solution in XQuery:

```
<bib>
 {
  FOR $b IN document("http://www.bn.com")/bib/book
  WHERE $b/publisher = "Addison-Wesley" AND $b/@year > 1991
  RETURN
          <book year={ $b/@year }>
           { $b/title }
          </book>
 }
</bib>
```

Expected Result:

```
<bib>
         <book year="1994">
                  <title>TCP/IP Illustrated</title>
         </book>
         <book year="1992">
                  <title>Advanced Programming in the Unix environment</title>
         </book>
</bib>
```

1.1.9.2 Q2

Create a flat list of all the title-author pairs, with each pair enclosed in a "result" element.

Solution in XQuery:

```
<results>
  {
         FOR $b IN document("http://www.bn.com")/bib/book,
             $t IN $b/title,
             $a IN $b/author
         RETURN
                  <result>
                           { $t }
                           { $a }
                  </result>
  }
</results>
```

Expected Result:

```
<results>
        <result>
                <title>TCP/IP Illustrated</title>
                <author>
                        <last>Stevens</last>
                        <first>W.</first>
                </author>
        </result>
        <result>
                <title>Advanced Programming in the Unix environment</title>
                <author>
                        <last>Stevens</last>
                        <first>W.</first>
                </author>
        </result>
        <result>
                <title>Data on the Web</title>
                <author>
                        <last>Abiteboul</last>
                        <first>Serge</first>
                </author>
        </result>
        <result>
                <title>Data on the Web</title>
                <author>
                        <last>Buneman</last>
                        <first>Peter</first>
                </author>
        </result>
        <result>
                <title>Data on the Web</title>
                <author>
                        <last>Suciu</last>
                        <first>Dan</first>
                </author>
        </result>
</results>
```

1.1.9.3 Q3

For each book in the bibliography, list the title and authors, grouped inside a "result" element.

Solution in XQuery:

```
<results>
  {
        FOR $b IN document("http://www.bn.com")/bib/book
        RETURN
                <result>
                        { $b/title }
                        {
                                FOR $a IN $b/author
                                RETURN $a
                        }
                </result>
  }
</results>
```

Expected Result:

```
<results>
         <result>
                  <title>TCP/IP Illustrated</title>
                  <author>
                           <last>Stevens</last>
                           <first>W.</first>
                  </author>
         </result>
         <result>
                  <title>Advanced Programming in the Unix environment</title>
                  <author>
                           <last>Stevens</last>
                           <first>W.</first>
                  </author>
         </result>
         <result>
                  <title>Data on the Web</title>
                  <author>
                           <last>Abiteboul</last>
                           <first>Serge</first>
                  </author>
                  <author>
                           <last>Buneman</last>
                           <first>Peter</first>
                  </author>
                  <author>
                           <last>Suciu</last>
                           <first>Dan</first>
                  </author>
         </result>
         <result>
                  <title>The Economics of Technology and Content for Digital TV</t
         </result>
</results>
```

1.1.9.4 Q4

For each author in the bibliography, list the author's name and the titles of all books by that author, grouped inside a "result" element.

Solution in XQuery:

```
<results>
  {
         FOR $a IN distinct(document("http://www.bn.com")//author)
         RETURN
                  <result>
                           { $a }
                           {
                                    FOR $b IN document("http://www.bn.com")/bib/bo
                                    RETURN $b/title
```

```
                        }
                </result>
  }
</results>
```

Expected Result:

```
<results>
        <result>
                <author>
                        <last>Stevens</last>
                        <first>W.</first>
                </author>
                <title>TCP/IP Illustrated</title>
                <title>Advanced Programming in the Unix environment</title>
        </result>
        <result>
                <author>
                        <last>Abiteboul</last>
                        <first>Serge</first>
                </author>
                <title>Data on the Web</title>
        </result>
        <result>
                <author>
                        <last>Buneman</last>
                        <first>Peter</first>
                </author>
                <title>Data on the Web</title>
        </result>
        <result>
                <author>
                        <last>Suciu</last>
                        <first>Dan</first>
                </author>
                <title>Data on the Web</title>
        </result>
</results>
```

1.1.9.5 Q5

For each book found at both bn.com and amazon.com, list the title of the book and its price from each source.

Solution in XQuery:

```
<books-with-prices>
  {
        FOR $b IN document("www.bn.com/bib.xml")//book,
            $a IN document("www.amazon.com/reviews.xml")//entry
        WHERE $b/title = $a/title
        RETURN
                <book-with-prices>
                        { $b/title }
```

```
                        <price-amazon>{ $a/price/text() }</price-amazon>
                        <price-bn>{ $b/price/text() }</price-bn>
                </book-with-prices>
  }
</books-with-prices>
```

Expected Result:

```
<books-with-prices>
        <book-with-prices>
                <title>TCP/IP Illustrated</title>
                <price-amazon>65.95</price-amazon>
                <price-bn> 65.95</price-bn>
        </book-with-prices>
        <book-with-prices>
                <title>Advanced Programming in the Unix environment</title>
                <price-amazon>65.95</price-amazon>
                <price-bn>65.95</price-bn>
        </book-with-prices>
        <book-with-prices>
                <title>Data on the Web</title>
                <price-amazon>34.95</price-amazon>
                <price-bn> 39.95</price-bn>
        </book-with-prices>
</books-with-prices>
```

1.1.9.6 Q6

For each book that has at least one author, list the title and first two authors, and an empty "et-al" element if the book has additional authors.

Solution in XQuery:

```
<bib>
  {
        FOR $b IN document("www.bn.com/bib.xml")//book
        WHERE count($b/author) > 0
        RETURN
                <book>
                        { $b/title }
                        {
                                FOR $a IN $b/author[1 TO 2]
                                RETURN $a
                        }
                        {
                                IF (count($b/author) > 2)
                                THEN <et-al/>
                                ELSE ()
                        }
                </book>
  }
</bib>
```

Expected Result:

```
<bib>
        <book>
                <title>TCP/IP Illustrated</title>
                <author>
                        <last>Stevens</last>
                        <first>W.</first>
                </author>
        </book>
        <book>
                <title>Advanced Programming in the Unix environment</title>
                <author>
                        <last>Stevens</last>
                        <first>W.</first>
                </author>
        </book>
        <book>
                <title>Data on the Web</title>
                <author>
                        <last>Abiteboul</last>
                        <first>Serge</first>
                </author>
                <author>
                        <last>Buneman</last>
                        <first>Peter</first>
                </author>
                <et-al/>
        </book>
</bib>
```

1.1.9.7 Q7

List the titles and years of all books published by Addison-Wesley after 1991, in alphabetic order.

Solution in XQuery:

```
<bib>
  {
        FOR $b IN document("www.bn.com/bib.xml")//book[publisher = "Addison-Wesley
        RETURN
                <book>
                        { $b/@year }
                        { $b/title }
                </book>
        SORTBY (title)
  }
</bib>
```

Expected Result:

```
<bib>
        <book year="1992">
                <title>Advanced Programming in the Unix environment</title>
        </book>
```

```
        <book year="1994">
                <title>TCP/IP Illustrated</title>
        </book>
</bib>
```

1.1.9.8 Q8

Find books in which some element has a tag ending in "or" and the same element contains the string "Suciu" (at any level of nesting). For each such book, return the title and the qualifying element.

Solution in XQuery:

```
FOR $b IN document("www.bn.com/bib.xml")//book,
    $e IN $b/*[contains(string(.), "Suciu")]
WHERE ends_with(name($e), "or")
RETURN
        <book>
                { $b/title }
                { $e }
        </book>
```

In the above solution, name(element) is an XPath function, and ends_with(string1, string2) is a possible function inspired by the XPath starts-with function, but not agreed on as an XQuery function.

Expected Result:

```
<book>
                <title>Data on the Web</title>
                <author>
                        <last>Suciu</last>
                        <first>Dan</first>
                </author>
 </book>
```

1.1.9.9 Q9

In the document "books.xml", find all section or chapter titles that contain the word "XML", regardless of the level of nesting.

Solution in XQuery:

```
<results>
  {
        FOR $t IN document("books.xml")//chapter/title UNION document("books.xml")
        WHERE contains($t/text(), "XML")
        RETURN $t
  }
</results>
```

Expected Result:

```
<results>
         <title>XML</title>
         <title>XML and Semistructured Data</title>
</results>
```

1.1.9.10 Q10

In the document "prices.xml", find the minimum price for each book, in the form of a "minprice" element with the book title as its title attribute.

Solution in XQuery:

```
<results>
  {
         LET $doc := document("prices.xml")
         FOR $t IN distinct($doc/book/title)
         LET $p := $doc/book[title = $t]/price
         RETURN
           <minprice title={ $t/text() }>
                   {
                     min($p)
                   }
           </minprice>
  }
</results>
```

Expected Result:

```
<results>
         <minprice title="Advanced Programming in the Unix environment">
                  <price>65.95</price>
         </minprice>
         <minprice title="TCP/IP Illustrated">
                  <price>65.95</price>
         </minprice>
         <minprice title="Data on the Web">
                  <price>39.95</price>
         </minprice>
</results>
```

1.1.9.11 Q11

For each book with an author, return the book with its title and authors. For each book with an editor, return a reference with the book title and the editor's affiliation.

Solution in XQuery:

```
<bib>
        {
                FOR $b IN document("www.bn.com/bib.xml")//book[author]
                RETURN
                        <book>
                                { $b/title }
                                { $b/author }
                        </book>
        }
        {
                FOR $b IN document("www.bn.com/bib.xml")//book[editor]
                RETURN
                        <reference>
                                { $b/title }
                                <org>{ $b/editor/affiliation/text() }</org>
                        </reference>
        }
</bib>
```

Expected Result:

```
<bib>
        <book>
                <title>TCP/IP Illustrated</title>
                <author>
                        <last>Stevens</last>
                        <first>W.</first>
                </author>
        </book>
        <book>
                <title>Advanced Programming in the Unix environment</title>
                <author>
                        <last>Stevens</last>
                        <first>W.</first>
                </author>
        </book>
        <book>
                <title>Data on the Web</title>
                <author>
                        <last>Abiteboul</last>
                        <first>Serge</first>
                </author>
                <author>
                        <last>Buneman</last>
                        <first>Peter</first>
                </author>
                <author>
                        <last>Suciu</last>
                        <first>Dan</first>
                </author>
        </book>
        <reference>
                <title>The Economics of Technology and Content for Digital TV</t
                <org>CITI</org>
        </reference>
</bib>
```

1.1.9.12 Q12

Find pairs of books that have different titles but the same set of authors (possibly in a different order).

Solution in XQuery:

```
<bib>
  {
         FOR $book1 IN document("www.bn.com/bib.xml")//book,
             $book2 IN document("www.bn.com/bib.xml")//book
         WHERE $book1/title/text() > $book2/title/text()
         AND bags-are-equal($book1/author, $book2/author)
         RETURN
                  <book-pair>
                           { $book1/title }
                           { $book2/title }
                  </book-pair>
  }
</bib>
```

Expected Result:

```
<bib>
       <book-pair>
              <title>TCP/IP Illustrated</title>
              <title>Advanced Programming in the Unix environment</title>
       </book-pair>
</bib>
```

Ed. Note: The above solution uses a function, bags-are-equal(), which was created out of thin air for the purpose of this query. The XQuery document contains an issue list, and one of the issues states that we must decide how to compare collections, including bags.

1.2 Use Case "TREE": Queries that preserve hierarchy

Some XML document-types have a very flexible structure in which text is mixed with elements and many elements are optional. These document-types show a wide variation in structure from one document to another. In documents of these types, the ways in which elements are ordered and nested are usually quite important.

1.2.1 Description

An XML query language should have the ability to extract elements from documents while preserving their original hierarchy. This Use Case illustrates this requirement by means of a flexible document type named Book.

1.2.2 Document Type Definition (DTD)

This use case is based on an input document named "book.xml", with the following DTD:

```
<!DOCTYPE book [
```

```
  <!ELEMENT book (title, author+, section+)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT section (title, (p | figure | section)* )>
  <!ATTLIST section
      id         ID    #IMPLIED
      difficulty CDATA #IMPLIED>
  <!ELEMENT p (#PCDATA)>
  <!ELEMENT figure (title, image)>
  <!ATTLIST figure
       width   CDATA   #REQUIRED
       height  CDATA   #REQUIRED >
  <!ELEMENT image EMPTY>
  <!ATTLIST image
       source  CDATA   #REQUIRED >
]>
```

1.2.3 Sample Data

The queries in this use case are based on the following sample data.

```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM "book.dtd">
<book>
  <title>Data on the Web</title>
  <author>Serge Abiteboul</author>
  <author>Peter Buneman</author>
  <author>Dan Suciu</author>
  <section id="intro" difficulty="easy" >
    <title>Introduction</title>
    <p>Text ... </p>
    <section>
      <title>Audience</title>
      <p>Text ... </p>
    </section>
    <section>
      <title>Web Data and the Two Cultures</title>
      <p>Text ... </p>
      <figure height="400" width="400">
        <title>Traditional client/server architecture</title>
        <image source="csarch.gif"/>
      </figure>
      <p>Text ... </p>
    </section>
  </section>
  <section id="syntax" difficulty="medium" >
    <title>A Syntax For Data</title>
    <p>Text ... </p>
    <figure height="200" width="500">
      <title>Graph representations of structures</title>
      <image source="graphs.gif"/>
    </figure>
    <p>Text ... </p>
    <section>
      <title>Base Types</title>
      <p>Text ... </p>
    </section>
    <section>
      <title>Representing Relational Databases</title>
```

```
      <p>Text ... </p>
      <figure height="250" width="400">
        <title>Examples of Relations</title>
        <image source="relations.gif"/>
      </figure>
    </section>
    <section>
      <title>Representing Object Databases</title>
      <p>Text ... </p>
    </section>
  </section>
</book>
```

1.2.4 Queries and Results

1.2.4.1 Q1

Prepare a (nested) table of contents for Book1, listing all the sections and their titles. Preserve the original attributes of each <section> element, if any.

Solution in XQuery:

```
<toc>
  {
        LET $b := document("book1.xml")
        RETURN
                filter($b//section | $b//section/title | $b//section/title/text(
  }
</toc>
```

Expected Result:

```
<toc>
        <section id="intro" difficulty="easy">
                <title>Introduction</title>
                <section>
                        <title>Audience</title>
                </section>
                <section>
                        <title>Web Data and the Two Cultures</title>
                </section>
        </section>
        <section id="syntax" difficulty="medium">
                <title>A Syntax For Data</title>
                <section>
                        <title>Base Types</title>
                </section>
                <section>
                        <title>Representing Relational Databases</title>
                </section>
                <section>
                        <title>Representing Object Databases</title>
                </section>
        </section>
</toc>
```

1.2.4.2 Q2

Prepare a (flat) figure list for Book1, listing all the figures and their titles. Preserve the original attributes of each <figure> element, if any.

Solution in XQuery:

```
<figlist>
  {
          FOR $f IN document("book1.xml")//figure
          RETURN
                    <figure>
                              { $f/@* }
                              { $f/title }
                    </figure>
  }
</figlist>
```

Expected Result:

```
<figlist>
          <figure  height="400" width="400">
                    <title>Traditional client/server architecture</title>
          </figure>
          <figure  height="200" width="500">
                    <title>Graph representations of structures</title>
          </figure>
          <figure  height="250" width ="400">
                    <title>Examples of Relations</title>
          </figure>
</figlist>
```

1.2.4.3 Q3

How many sections are in Book1, and how many figures?

Solution in XQuery:

```
<section_count>{ count(document("book1.xml")//section) }</section_count>,
<figure_count>{ count(document("book1.xml")//figure) }</figure_count>
```

Expected Result:

```
<section_count>7</section_count>
<figure_count>3</figure_count>
```

1.2.4.4 Q4

How many top-level sections are in Book1?

Solution in XQuery:

```
<top_section_count>
 {
   count(document("book1.xml")/book/section)
 }
</top_section_count>
```

Expected Result:

```
<top_section_count>2</top_section_count>
```

1.2.4.5 Q5

Make a flat list of the section elements in Book1. In place of its original attributes, each section element should have two attributes, containing the title of the section and the number of figures immediately contained in the section.

Solution in XQuery:

```
<section_list>
  {
         FOR $s IN document("book1.xml")//section
         LET $f := $s/figure
         RETURN
                  <section title={ $s/title/text() } figcount={ count($f) }/>
  }
</section_list>
```

Expected Result:

```
<section_list>
         <section title="Introduction" figcount="0"/>
         <section title="A Syntax For Data" figcount="1"/>
         <section title="Audience" figcount="0"/>
         <section title="Web Data and the Two Cultures" figcount="1"/>
         <section title="Base Types" figcount="0"/>
         <section title="Representing Relational Databases" figcount="1"/>
         <section title="Representing Object Databases" figcount="0"/>
</section_list>
```

1.2.4.6 Q6

Make a nested list of the section elements in Book1, preserving their original attributes and hierarchy. Inside each section element, include the title of the section and an element that includes the number of figures immediately contained in the section.

Solution in XQuery:

```
DEFINE FUNCTION section_summary (ELEMENT $s) RETURNS ELEMENT
{
         <section>
                  { $s/@* }
                  { $s/title }
                  <figcount>{ count($s/figure) }</figcount>
                  {
                           FOR $ss IN $s/section
                           RETURN section_summary($ss)
                  }
         </section>
}

<toc>
  {
         FOR $s IN document("book1.xmll")//section
         RETURN section_summary($s)
  }
</toc>
```

Expected Result:

```
<toc>
         <section>
                  <title>Introduction</title>
                  <figcount>0</figcount>
                  <section>
                           <title>Audience</title>
                           <figcount>0</figcount>
                  </section>
                  <section>
                           <title>Web Data and the Two Cultures</title>
                           <figcount>1</figcount>
                  </section>
         </section>
         <section>
                  <title>A Syntax For Data</title>
                  <figcount>1</figcount>
                  <section>
                           <title>Base Types</title>
                           <figcount>0</figcount>
                  </section>
                  <section>
                           <title>Representing Relational Databases</title>
                           <figcount>1</figcount>
                  </section>
                  <section>
                           <title>Representing Object Databases</title>
                           <figcount>0</figcount>
                  </section>
         </section>
         <section>
                  <title>Audience</title>
```

```
                <figcount>0</figcount>
            </section>
            <section>
                <title>Web Data and the Two Cultures</title>
                <figcount>1</figcount>
            </section>
            <section>
                <title>Base Types</title>
                <figcount>0</figcount>
            </section>
            <section>
                <title>Representing Relational Databases</title>
                <figcount>1</figcount>
            </section>
            <section>
                <title>Representing Object Databases</title>
                <figcount>0</figcount>
            </section>
</toc>
```

1.3 Use Case "SEQ" - Queries based on Sequence

This use case illustrates queries based on the sequence in which elements appear in a document.

1.3.1 Description

Although sequence is not significant in most traditional database systems or object systems, it can be quite significant in structured documents. This use case presents a series of queries based on a medical report.

1.3.2 Document Type Definition (DTD)

This use case is based on a medical report using the HL7 Patient Record Architecture. We simplify the DTD in this example, using only what is needed to understand the queries.

```
<!DOCTYPE report [
  <!ELEMENT report (section*)>
  <!ELEMENT section (section.title, section.content)>
  <!ELEMENT section.title  (#PCDATA )>
  <!ELEMENT section.content  (#PCDATA | anesthesia | prep
                            | incision | action | observation )*>
  <!ELEMENT anesthesia (#PCDATA)>
  <!ELEMENT prep ( (#PCDATA | action)* )>
  <!ELEMENT incision ( (#PCDATA | geography | instrument)* )>
  <!ELEMENT action ( (#PCDATA | instrument )* )>
  <!ELEMENT observation (#PCDATA)>
  <!ELEMENT geography (#PCDATA)>
  <!ELEMENT instrument (#PCDATA)>
]>
```

1.3.3 Sample Data

The queries in this use case are based on the following sample data.

```
<report>
  <section>
    <section.title>Procedure</section.title>
```

```
            <section.content>
          The patient was taken to the operating room where she was placed
          in supine position and
          <anesthesia>induced under general anesthesia.</anesthesia>
          <prep>
            <action>A Foley catheter was placed to decompress the bladder</action>
            and the abdomen was then prepped and draped in sterile fashion.
          </prep>
          <incision>
            A curvilinear incision was made
            <geography>in the midline immediately infraumbilical</geography>
            and the subcutaneous tissue was divided
            <instrument>using electrocautery.</instrument>
          </incision>
          The fascia was identified and
          <action>#2 0 Maxon stay sutures were placed on each side of the midline.</act
          <incision>
            The fascia was divided using
            <instrument>electrocautery</instrument>
            and the peritoneum was entered.
          </incision>
          <observation>The small bowel was identified.</observation>
          and
          <action>
            the
            <instrument>Hasson trocar</instrument>
            was placed under direct visualization.
          </action>
          <action>
            The
            <instrument>trocar</instrument>
            was secured to the fascia using the stay sutures.
          </action>
            </section.content>
    </section>
  </report>
```

1.3.4 Queries and Results

1.3.4.1 Q1

In the Procedure section of Report1, what Instruments were used in the second Incision?

Solution in XQuery:

```
FOR $s IN document("report1.xml")//section[section.title = "Procedure"]
RETURN ($s//incision)[2]/instrument
```

Expected Result:

```
<instrument>electrocautery</instrument>
```

1.3.4.2 Q2

In the Procedure section of Report1, what are the first two Instruments to be used?

Solution in XQuery:

```
FOR $s IN document("report1.xml")//section[section.title = "Procedure"]
RETURN ($s//instrument)[1 TO 2]
```

Expected Result:

```
<instrument>using electrocautery.</instrument>
<instrument>electrocautery</instrument>
```

1.3.4.3 Q3

In Report1, what Instruments were used in the first two Actions after the second Incision?

Solution in XQuery:

```
LET $i2 := (document("report1.xml")//incision)[2]
FOR $a IN (document("report1.xml")//action AFTER $i2)[1 TO 2]
RETURN $a//instrument
```

Expected Result:

```
<instrument>Hasson trocar</instrument>
<instrument>trocar</instrument>
```

1.3.4.4 Q4

In Report1, find "Procedure" sections where no Anesthesia element occurs before the first Incision

Solution in XQuery:

```
FOR $proc IN document("data/seq-data.xml")//section[section.title = "Procedure"]
WHERE SOME $i IN $proc//incision SATISFIES empty($proc//anesthesia BEFORE $i)
RETURN $proc
```

Expected Result:

```
<result />
```

(No sections satisfy Q4, thankfully.)

1.3.4.5 Q5

In Report1, what happened between the first Incision and the second Incision?

Solution in XQuery:

```
FOR $proc IN document("data/seq-data.xml")//section[section.title = "Procedure"]
RETURN ($proc//* AFTER ($proc//incision)[1]) BEFORE ($proc//incision)[2]
```

Expected Result:

```
<action>#2 0 Maxon stay sutures were placed on each side of the midline.</action>
```

1.4 Use Case "R" - Access to Relational Data

One important use of an XML query language will be to access data stored in relational databases. This use case describes one possible way in which this access might be accomplished.

1.4.1 Description

A relational database system might present a view in which each table (relation) takes the form of an XML document. One way to represent a database table as an XML document is to allow the document element to represent the table itself, and each row (tuple) inside the table to be represented by a nested element. Inside the tuple-elements, each column is in turn represented by a nested element. Columns that allow null values are represented by optional elements, and a missing element denotes a null value.

As an example, consider a relational database used by an online auction. The auction maintains a USERS table containing information on registered users, each identified by a unique userid, who can either offer items for sale or bid on items. An ITEMS table lists items currently or recently for sale, with the userid of the user who offered each item. A BIDS table contains all bids on record, keyed by the userid of the bidder and the item number of the item to which the bid applies.

The three tables used by the online auction are below, with their column-names indicated in parentheses.

```
USERS ( USERID, NAME, RATING )

ITEMS ( ITEMNO, DESCRIPTION, OFFERED_BY, START_DATE, END_DATE, RESERVE_PRICE )

BIDS ( USERID, ITEMNO, BID, BID_DATE )
```

1.4.2 Document Type Definition (DTD)

This use case is based on three separate input documents named users.xml, items.xml, and bids.xml. Each of the documents represents one of the tables in the relational database described above, using the following DTDs:

```
<!DOCTYPE users [
  <!ELEMENT users   (user_tuple*)>
  <!ELEMENT user_tuple (userid, name, rating?)>
  <!ELEMENT userid  (#PCDATA)>
  <!ELEMENT name    (#PCDATA)>
  <!ELEMENT rating  (#PCDATA)>
]>

<!DOCTYPE items [
  <!ELEMENT items       (item_tuple*)>
  <!ELEMENT item_tuple  (itemno, description, offered_by,
                         start_date?, end_date?, reserve_price? )>
  <!ELEMENT itemno      (#PCDATA)>
  <!ELEMENT description (#PCDATA)>
  <!ELEMENT offered_by  (#PCDATA)>
  <!ELEMENT start_date  (#PCDATA)>
  <!ELEMENT end_date    (#PCDATA)>
  <!ELEMENT reserve_price (#PCDATA)>
]>

<!DOCTYPE bids [
  <!ELEMENT bids      (bid_tuple*)>
  <!ELEMENT bid_tuple (userid, itemno, bid, bid_date)>
  <!ELEMENT userid    (#PCDATA)>
  <!ELEMENT itemno    (#PCDATA)>
  <!ELEMENT bid       (#PCDATA)>
  <!ELEMENT bid_date  (#PCDATA)>
]>
```

1.4.3 Sample Data

The following tables contain example data. Representation of this data in the DTD of this Use Case is straightforward but rather bulky, and is not included here.

USERS

USERID	NAME	RATING
U01	Tom Jones	B
U02	Mary Doe	A
U03	Dee Linquent	D
U04	Roger Smith	C
U05	Jack Sprat	B
U06	Rip Van Winkle	B

ITEMS

ITEMNO	DESCRIPTION	OFFERED_BY	START_DATE	END_DATE	RESERVE_PRICE
1001	Red Bicycle	U01	99-01-05	99-01-20	40
1002	Motorcycle	U02	99-02-11	99-03-15	500
1003	Old Bicycle	U02	99-01-10	99-02-20	25
1004	Tricycle	U01	99-02-25	99-03-08	15
1005	Tennis Racket	U03	99-03-19	99-04-30	20
1006	Helicopter	U03	99-05-05	99-05-25	50000

1007	Racing Bicycle	U04	99-01-20	99-02-20	200
1008	Broken Bicycle	U01	99-02-05	99-03-06	25

BIDS

USERID	ITEMNO	BID	BID_DATE
U02	1001	35	99-01-07
U04	1001	40	99-01-08
U02	1001	45	99-01-11
U04	1001	50	99-01-13
U02	1001	55	99-01-15
U01	1002	400	99-02-14
U02	1002	600	99-02-16
U03	1002	800	99-02-17
U04	1002	1000	99-02-25
U02	1002	1200	99-03-02
U04	1003	15	99-01-22
U05	1003	20	99-02-03
U01	1004	40	99-03-05
U03	1007	175	99-01-25
U05	1007	200	99-02-08
U04	1007	225	99-02-12

1.4.4 Queries and Results

The following results assume that the queries were executed on Feb. 1, 1999.

1.4.4.1 Q1

List the item number and description of all bicycles that currently have an auction in progress, ordered by item number.

Solution in XQuery:

```
<result>
  {
        FOR $i IN document("items.xml")//item_tuple
        WHERE $i/start_date <= date()
          AND $i/end_date >= date()
          AND contains($i/description, "Bicycle")
        RETURN
                <item_tuple>
                        { $i/itemno }
                        { $i/description }
                </item_tuple>
        SORTBY (itemno)
  }
</result>
```

Expected Result:

```
<result>
         <item_tuple>
                  <itemno>1003</itemno>
                  <description>Old Bicycle</description>
                  <high_bid>
                           <bid>20</bid>
                  </high_bid>
         </item_tuple>
         <item_tuple>
                  <itemno>1007</itemno>
                  <description>Racing Bicycle</description>
                  <high_bid>
                           <bid>225</bid>
                  </high_bid>
         </item_tuple>
</result>
```

1.4.4.2 Q2

For all bicycles, list the item number, description, and highest bid (if any), ordered by item number.

Solution in XQuery:

```
<result>
  {
         FOR $i IN document("items.xml")//item_tuple
         LET $b := document("bids.xml")//bid_tuple[itemno = $i/itemno]
         WHERE contains($i/description, "Bicycle")
         RETURN
                  <item_tuple>
                           { $i/itemno }
                           { $i/description }
                           <high_bid>{ max($b/bid) }</high_bid>
                  </item_tuple>
         SORTBY(itemno)
  }
</result>
```

Expected Result:

```
<result>
         <item_tuple>
                  <itemno>1001</itemno>
                  <description>Red Bicycle</description>
                  <high_bid>
                           <bid>55</bid>
                  </high_bid>
         </item_tuple>
```

```
        <item_tuple>
                <itemno>1003</itemno>
                <description>Old Bicycle</description>
                <high_bid>
                        <bid>20</bid>
                </high_bid>
        </item_tuple>
        <item_tuple>
                <itemno>1007</itemno>
                <description>Racing Bicycle</description>
                <high_bid>
                        <bid>225</bid>
                </high_bid>
        </item_tuple>
        <item_tuple>
                <itemno>1008</itemno>
                <description>Broken Bicycle</description>
                <high_bid/>
        </item_tuple>
</result>
```

1.4.4.3 Q3

Find cases where a user with a rating worse (alphabetically, greater) than "C" is offering an item with a reserve price of more than 1000.

Solution in XQuery:

```
<result>
  {
        FOR $u IN document("users.xml")//user_tuple
        FOR $i IN document("items.xml")//item_tuple
        WHERE $u/rating > "C"
           AND $i/reserve_price > 1000
           AND $i/offered_by = $u/userid
        RETURN
                <warning>
                        { $u/name }
                        { $u/rating }
                        { $i/description }
                        { $i/reserve_price }
                </warning>
  }
</result>
```

Expected Result:

```
<result>
        <warning>
                <name>Dee Linquent</name>
                <rating>D</rating>
                <description>Helicopter</description>
                <reserve_price>50000</reserve_price>
        </warning>
</result>
```

1.4.4.4 Q4

List item numbers and descriptions of items that have no bids.

Solution in XQuery:

```
<result>
  {
         FOR $i IN document("items.xml")//item_tuple
         WHERE not(SOME $b IN document("bids.xml")//bid_tuple
                                  SATISFIES $b/itemno = $i/itemno)
         RETURN
                  <no_bid_item>
                           { $i/itemno }
                           { $i/description }
                  </no_bid_item>
  }
</result>
```

Expected Result:

```
<result>
         <no_bid_item>
                  <itemno>1005</itemno>
                  <description>Tennis Racket</description>
         </no_bid_item>
         <no_bid_item>
                  <itemno>1006</itemno>
                  <description>Helicopter</description>
         </no_bid_item>
         <no_bid_item>
                  <itemno>1008</itemno>
                  <description>Broken Bicycle</description>
         </no_bid_item>
</result>
```

1.4.4.5 Q5

For bicycle(s) offered by Tom Jones, list the item number, description, highest bid (if any), and name of the highest bidder, ordered by item number.

Solution in XQuery:

```
<result>
  {
         FOR $seller IN document("users.xml")//user_tuple,
             $buyer IN  document("users.xml")//user_tuple,
             $item IN  document("items.xml")//item_tuple,
             $highbid IN  document("bids.xml")//bid_tuple
         WHERE $seller/name = "Tom Jones"
```

```
            AND $seller/userid = $item/offered_by
            AND contains($item/description, "Bicycle")
            AND $item/itemno = $highbid/itemno
            AND $highbid/userid = $buyer/userid
            AND $highbid/bid = max(document("bids.xml")//bid_tuple [itemno = $item/i
          RETURN
                   <jones_bike>
                            { $item/itemno }
                            { $item/description }
                            <high_bid>{ $highbid/bid }</high_bid>
                            <high_bidder>{ $buyer/name }</high_bidder>
                   </jones_bike>
          SORTBY(itemno)
  }
</result>
```

The following alternate solution uses the `unordered` function to inform the query processor that the order of the lists in the FOR clause is not significant, which means that the tuples can be generated in any order. This can enable better optimization.

Alternate Solution in XQuery:

```
<result>
  {
          FOR $seller IN unordered(document("users.xml")//user_tuple)),
              $buyer IN  unordered(document("users.xml")//user_tuple),
              $item IN  unordered(document("items.xml")//item_tuple),
              $highbid IN  unordered(document("bids.xml")//bid_tuple)
          WHERE $seller/name = "Tom Jones"
            AND $seller/userid = $item/offered_by
            AND contains($item/description, "Bicycle")
            AND $item/itemno = $highbid/itemno
            AND $highbid/userid = $buyer/userid
            AND $highbid/bid = max(document("bids.xml")//bid_tuple [itemno = $item/i
          RETURN
                   <jones_bike>
                            { $item/itemno }
                            { $item/description }
                            <high_bid>{ $highbid/bid }</high_bid>
                            <high_bidder>{ $buyer/name }</high_bidder>
                   </jones_bike>
          SORTBY(itemno)
  }
</result>
```

Expected Result:

```
<result>
          <jones_bike>
                   <itemno>1001</itemno>
                   <description>Red Bicycle</description>
                   <high_bid>
                            <bid>55</bid>
                   </high_bid>
```

```
                <high_bidder>
                        <name>Mary Doe</name>
                </high_bidder>
        </jones_bike>
</result>
```

1.4.4.6 Q6

For each item whose highest bid is more than twice its reserve price, list the item number, description, reserve price, and highest bid.

Solution in XQuery:

```
<result>
  {
        FOR $item IN document("items.xml")//item_tuple
        LET $b := document("bids.xml")//bid_tuple[itemno = $item/itemno]
        WHERE $item/reserve_price * 2 < max($b/bid)
        RETURN
                <successful_item>
                        { $item/itemno }
                        { $item/description }
                        { $item/reserve_price }
                        <high_bid>{ max($b/bid) }</high_bid>
                </successful_item>
 }
</result>
```

Expected Result:

```
<result>
        <successful_item>
                <itemno>1002</itemno>
                <description>Motorcycle</description>
                <reserve_price>500</reserve_price>
                <high_bid>
                        <bid>1200</bid>
                </high_bid>
        </successful_item>
        <successful_item>
                <itemno>1004</itemno>
                <description>Tricycle</description>
                <reserve_price>15</reserve_price>
                <high_bid>
                        <bid>40</bid>
                </high_bid>
        </successful_item>
</result>
```

1.4.4.7 Q7

Find the highest bid ever made for a bicycle or tricycle.

Solution in XQuery:

```
LET $allbikes := document("items.xml")//item_tuple[contains(description, "Bicycle")
LET $bikebids := document("bids.xml")//bid_tuple[itemno = $allbikes/itemno]
RETURN
         <high_bid>
           {
             max($bikebids/bid)
           }
         </high_bid>
```

Expected Result:

```
<high_bid><bid>225</bid></high_bid>
```

1.4.4.8 Q8

How many items were actioned (auction ended) in March 1999?

Solution in XQuery:

```
LET $item := document("items.xml")//item_tuple
  [end_date >= date("1999-03-01") AND end_date <= date("1999-03-31")]
RETURN
         <item_count>
           {
             count($item)
           }
         </item_count>
```

Expected Result:

```
<item_count>3</item_count>
```

1.4.4.9 Q9

List the number of items auctioned each month in 1999 for which data is available, ordered by month.

Solution in XQuery:

```
<result>
  {
         LET $end_dates := document("items.xml")//item_tuple/end_date
         FOR $m IN distinct(month($end_dates))
         LET $item := document("items.xml")
             //item_tuple[year(end_date) = 1999 AND month(end_date) = $m]
```

```
        RETURN
                <monthly_result>
                        <month>{ $m }</month>
                        <item_count>{ count($item) }</item_count>
                </monthly_result>
        SORTBY(month)
  }
</result>
```

Expected Result:

```
<result>
  <monthly_result>
    <month> 1 </month>
    <item_count> 1 </item_count>
  </monthly_result>
  <monthly_result>
    <month> 2 </month>
    <item_count> 2 </item_count>
  </monthly_result>
  <monthly_result>
    <month> 3 </month>
    <item_count> 3 </item_count>
  </monthly_result>
  <monthly_result>
    <month> 4 </month>
    <item_count> 1 </item_count>
  </monthly_result>
  <monthly_result>
    <month> 5 </month>
    <item_count> 1 </item_count>
  </monthly_result>
</result>
```

1.4.4.10 Q10

For each item that has received a bid, list the item number, the highest bid, and the name of the highest bidder, ordered by item number.

Solution in XQuery:

```
<result>
  {
        FOR $highbid IN document("bids.xml")//bid_tuple,
            $user IN document("users.xml")//user_tuple
        WHERE $user/userid = $highbid/userid
          AND $highbid/bid = max(document("bids.xml")//bid_tuple[itemno = $highbi
        RETURN
                <high_bid>
                        { $highbid/itemno }
                        { $highbid/bid }
                        <bidder>{ $user/name/text() }</bidder>
                </high_bid>
        SORTBY(itemno)
```

```
  }
</result>
```

Expected Result:

```
<result>
         <high_bid>
                  <itemno>1001</itemno>
                  <bid>55</bid>
                  <bidder>Mary Doe</bidder>
         </high_bid>
         <high_bid>
                  <itemno>1002</itemno>
                  <bid>1200</bid>
                  <bidder>Mary Doe</bidder>
         </high_bid>
         <high_bid>
                  <itemno>1003</itemno>
                  <bid>20</bid>
                  <bidder>Jack Sprat</bidder>
         </high_bid>
         <high_bid>
                  <itemno>1004</itemno>
                  <bid>40</bid>
                  <bidder>Tom Jones</bidder>
         </high_bid>
         <high_bid>
                  <itemno>1007</itemno>
                  <bid>225</bid>
                  <bidder>Roger Smith</bidder>
         </high_bid>
</result>
```

1.4.4.11 Q11

List the item number and description of the item(s) that received the highest bid ever recorded, and the amount of that bid.

Solution in XQuery:

```
LET $highbid := max(document("bids.xml")//bid_tuple/bid)
RETURN
         <result>
           {
                  FOR $item IN document("items.xml")//item_tuple,
                      $b IN document("bids.xml")//bid_tuple[itemno = $item/itemno]
                  WHERE $b/bid = $highbid
                  RETURN
                           <expensive_item>
                                    { $item/itemno }
                                    { $item/description }
                                    <high_bid>{ $highbid }</high_bid>
                           </expensive_item>
           }
```

```
        </result>
```

Expected Result:

```
<result>
        <expensive_item>
                <itemno>1002</itemno>
                <description>Motorcycle</description>
                <high_bid>
                        <bid>1200</bid>
                </high_bid>
        </expensive_item>
</result>
```

1.4.4.12 Q12

List the item number and description of the item(s) that received the largest number of bids, and the number of bids it (or they) received.

Solution in XQuery:

```
DEFINE FUNCTION bid_summary ()
{
        FOR $i IN distinct(document("bids.xml")//itemno)
        LET $b := document("bids.xml")//bid_tuple[itemno = $i]
        RETURN
                <bid_count>
                        { $i }
                        <nbids>{ count($b) }</nbids>
                </bid_count>
}

<result>
 {
        LET $bid_counts := bid_summary(),
            $maxbids := max($bid_counts/nbids),
            $maxitemnos := $bid_counts[nbids=$max_bids]
        FOR $item IN document("items.xml")//item_tuple,
            $bc IN $bid_counts
        WHERE $bc/nbids = $maxbids AND $item/itemno = $bc/itemno
        RETURN
                <popular_item>
                        { $item/itemno }
                        { $item/description }
                        <bid_count>{ $bc/nbids/text() }</bid_count>
                </popular_item>
 }
</result>
```

Expected Result:

```
<result>
         <popular_item>
                  <itemno>1001</itemno>
                  <description>Red Bicycle</description>
                  <bid_count>5</bid_count>
         </popular_item>
         <popular_item>
                  <itemno>1002</itemno>
                  <description>Motorcycle</description>
                  <bid_count>5</bid_count>
         </popular_item>
</result>
```

1.4.4.13 Q13

For each user who has placed a bid, give the userid, name, number of bids, and average bid, in order by userid.

Solution in XQuery:

```
<result>
  {
         FOR $uid IN distinct(document("bids.xml")//userid),
             $u IN document("users.xml")//user_tuple[userid = $uid]
         LET $b := document("bids.xml")//bid_tuple[userid = $uid]
         RETURN
                  <bidder>
                           { $u/userid }
                           { $u/name }
                           <bidcount>{ count($b) }</bidcount>
                           <avgbid>{ avg($b/bid) }</avgbid>
                  </bidder>
         SORTBY(userid)
  }
</result>
```

Expected Result:

```
<result>
         <bidder>
                  <userid>U01</userid>
                  <name>Tom Jones</name>
                  <bidcount>2</bidcount>
                  <avgbid>220</avgbid>
         </bidder>
         <bidder>
                  <userid>U02</userid>
                  <name>Mary Doe</name>
                  <bidcount>5</bidcount>
                  <avgbid>387</avgbid>
         </bidder>
         <bidder>
                  <userid>U03</userid>
                  <name>Dee Linquent</name>
```

```
                <bidcount>2</bidcount>
                <avgbid>487</avgbid>
        </bidder>
        <bidder>
                <userid>U04</userid>
                <name>Roger Smith</name>
                <bidcount>5</bidcount>
                <avgbid>266</avgbid>
        </bidder>
        <bidder>
                <userid>U05</userid>
                <name>Jack Sprat</name>
                <bidcount>2</bidcount>
                <avgbid>110</avgbid>
        </bidder>
</result>
```

1.4.4.14 Q14

List item numbers and average bids for items that have received three or more bids, in descending order by average bid.

Solution in XQuery:

```
<result>
  {
        FOR $i IN distinct(document("bids.xml")//itemno)
        LET $b := document("bids.xml")//bid_tuple[itemno = $i]
        WHERE count($b) >= 3
        RETURN
                <popular_item>
                        { $i }
                        <avgbid>{ avg($b/bid) }</avgbid>
                </popular_item>
        SORTBY(avgbid DESCENDING)
  }
</result>
```

Expected Result:

```
<result>
        <popular_item>
                <itemno>1002</itemno>
                <avgbid>800</avgbid>
        </popular_item>
        <popular_item>
                <itemno>1007</itemno>
                <avgbid>200</avgbid>
        </popular_item>
        <popular_item>
                <itemno>1001</itemno>
                <avgbid>45</avgbid>
        </popular_item>
</result>
```

1.4.4.15 Q15

List names of users who have placed multiple bids of at least $100 each.

Solution in XQuery:

```
<result>
  {
         FOR $u IN document("users.xml")//user_tuple
         LET $b := document("bids.xml")//bid_tuple[userid = $u/userid AND bid >= 10
         WHERE count($b) > 1
         RETURN
                  <big_spender>{ $u/name/text() }</big_spender>
  }
</result>
```

Expected Result:

```
<result>
         <big_spender>Mary Doe</big_spender>
         <big_spender>Dee Linquent</big_spender>
         <big_spender>Roger Smith</big_spender>
</result>
```

1.4.4.16 Q16

List all registered users in order by userid; for each user, include the userid, name, and an indication of whether the user is active (has at least one bid on record) or inactive (has no bid on record).

Solution in XQuery:

```
<result>
  {
         FOR $u IN document("users.xml")//user_tuple
         LET $b := document("bids.xml")//bid_tuple[userid = $u/userid]
         RETURN
                  <user>
                           { $u/userid }
                           { $u/name }
                           {
                                    IF (empty($b))
                                    THEN <status>inactive</status>
                                    ELSE <status>active</status>
                           }
                  </user>
         SORTBY(userid)
  }
</result>
```

Expected Result:

```
<result>
         <user>
                  <userid>U01</userid>
                  <name>Tom Jones</name>
                  <status>active</status>
         </user>
         <user>
                  <userid>U02</userid>
                  <name>Mary Doe</name>
                  <status>active</status>
         </user>
         <user>
                  <userid>U03</userid>
                  <name>Dee Linquent</name>
                  <status>active</status>
         </user>
         <user>
                  <userid>U04</userid>
                  <name>Roger Smith</name>
                  <status>active</status>
         </user>
         <user>
                  <userid>U05</userid>
                  <name>Jack Sprat</name>
                  <status>active</status>
         </user>
         <user>
                  <userid>U06</userid>
                  <name>Rip Van Winkle</name>
                  <status>inactive</status>
         </user>
</result>
```

1.4.4.17 Q17

List the names of users, if any, who have bid on every item.

Solution in XQuery:

```
<frequent_bidder>
  {
         FOR $u IN document("users.xml")//user_tuple
         WHERE
           EVERY $item IN document("items.xml")//item_tuple SATISFIES
             SOME $b IN document("bids.xml")//bid_tuple SATISFIES
               ($item/itemno = $b/itemno AND $u/userid = $b/userid)
         RETURN
                  $u/name
  }
</frequent_bidder>
```

Expected Result:

```
<frequent_bidder />
```

(No users satisfy Q17.)

1.4.4.18 Q18

List all users in alphabetic order by name. For each user, include descriptions of all the items (if any) that were bid on by that user, in alphabetic order.

Solution in XQuery:

```
<result>
  {
         FOR $u IN document("users.xml")//user_tuple
         RETURN
                  <user>
                           { $u/name }
                           {
                                    FOR $b IN distinct(document("bids.xml")//bid_t
                                        $i IN document("items.xml")//item_tuple[it
                                    RETURN
                                             <bid_on_item>{ $i/description/text()
                                    SORTBY(.)
                           }
                  </user>
         SORTBY(name)
  }
</result>
```

Expected Result:

```
<result>
         <user>
                  <name>Dee Linquent</name>
                  <bid_on_item>Motorcycle</bid_on_item>
                  <bid_on_item>Racing Bicycle</bid_on_item>
         </user>
         <user>
                  <name>Jack Sprat</name>
                  <bid_on_item>Old Bicycle</bid_on_item>
                  <bid_on_item>Racing Bicycle</bid_on_item>
         </user>
         <user>
                  <name>Mary Doe</name>
                  <bid_on_item>Motorcycle</bid_on_item>
                  <bid_on_item>Red Bicycle</bid_on_item>
         </user>
         <user>
                  <name>Rip Van Winkle</name>
         </user>
         <user>
```

```
                    <name>Roger Smith</name>
                    <bid_on_item>Motorcycle</bid_on_item>
                    <bid_on_item>Old Bicycle</bid_on_item>
                    <bid_on_item>Racing Bicycle</bid_on_item>
                    <bid_on_item>Red Bicycle</bid_on_item>
          </user>
          <user>
                    <name>Tom Jones</name>
                    <bid_on_item>Motorcycle</bid_on_item>
                    <bid_on_item>Tricycle</bid_on_item>
          </user>
</result>
```

1.5 Use Case "SGML": Standard Generalized Markup Language

1.5.1 Description

The example document and queries in this Use Case were first created for a 1992 conference on Standard Generalized Markup Language (SGML). For our use, the Document Type Definition (DTD) and example document have been translated from SGML to XML.

1.5.2 Document Type Definition (DTD)

This use case is based on an implicit (unnamed) input data set, using the DTD shown below.

```
<!NOTATION cgm PUBLIC "Computer Graphics Metafile">
<!NOTATION ccitt PUBLIC "CCITT group 4 raster">

<!ENTITY % text "(#PCDATA | emph)*">
<!ENTITY infoflow SYSTEM "infoflow.ccitt" NDATA ccitt>
<!ENTITY tagexamp SYSTEM "tagexamp.cgm"   NDATA cgm>

<!ELEMENT report (title, chapter+)>
<!ELEMENT title %text;>
<!ELEMENT chapter (title, intro?, section*)>
<!ATTLIST chapter
              shorttitle CDATA #IMPLIED>
<!ELEMENT intro (para | graphic)+>
<!ELEMENT section (title, intro?, topic*)>
<!ATTLIST section
              shorttitle CDATA #IMPLIED
              sectid ID #IMPLIED>
<!ELEMENT topic (title, (para | graphic)+)>
<!ATTLIST topic
              shorttitle CDATA #IMPLIED
              topicid ID #IMPLIED>
<!ELEMENT para (#PCDATA | emph | xref)*>
<!ATTLIST para
              security (u | c | s | ts) "u">
<!ELEMENT emph %text;>
<!ELEMENT graphic EMPTY>
<!ATTLIST graphic
              graphname ENTITY #REQUIRED>
<!ELEMENT xref EMPTY>
<!ATTLIST xref
              xrefid IDREF #IMPLIED>
```

1.5.3 Sample Data

The queries in this use case are based on the following sample data. Line numbers have been added to the data to allow the results of queries to be conveniently specified.

```
<?xml version="1.0"?>
<!-- 0--> <!DOCTYPE report SYSTEM "report.dtd">
<!-- 1--> <report>
<!-- 2--> <title>Getting started with SGML</title>
<!-- 3--> <chapter>
<!-- 4--> <title>The business challenge</title>
<!-- 5--> <intro>
<!-- 6--> <para>With the ever-changing and growing global market, companies and
<!-- 7--> large organizations are searching for ways to become more viable and
<!-- 8--> competitive. Downsizing and other cost-cutting measures demand more
<!-- 9--> efficient use of corporate resources. One very important resource is
<!--10--> an organization's information.</para>
<!--11--> <para>As part of the move toward integrated information management,
<!--12--> whole industries are developing and implementing standards for
<!--13--> exchanging technical information. This report describes how one such
<!--14--> standard, the Standard Generalized Markup Language (SGML), works as
<!--15--> part of an overall information management strategy.</para>
<!--16--> <graphic graphname="infoflow"/></intro></chapter>
<!--17--> <chapter>
<!--18--> <title>Getting to know SGML</title>
<!--19--> <intro>
<!--20--> <para>While SGML is a fairly recent technology, the use of
<!--21--> <emph>markup</emph> in computer-generated documents has existed for a
<!--22--> while.</para></intro>
<!--23--> <section shorttitle="What is markup?">
<!--24--> <title>What is markup, or everything you always wanted to know about
<!--25--> document preparation but were afraid to ask?</title>
<!--26--> <intro>
<!--27--> <para>Markup is everything in a document that is not content. The
<!--28--> traditional meaning of markup is the manual <emph>marking</emph> up
<!--29--> of typewritten text to give instructions for a typesetter or
<!--30--> compositor about how to fit the text on a page and what typefaces to
<!--31--> use. This kind of markup is known as <emph>procedural markup</emph>.</par
<!--32--> <topic topicid="top1">
<!--33--> <title>Procedural markup</title>
<!--34--> <para>Most electronic publishing systems today use some form of
<!--35--> procedural markup. Procedural markup codes are good for one
<!--36--> presentation of the information.</para></topic>
<!--37--> <topic topicid="top2">
<!--38--> <title>Generic markup</title>
<!--39--> <para>Generic markup (also known as descriptive markup) describes the
<!--40--> <emph>purpose</emph> of the text in a document. A basic concept of
<!--41--> generic markup is that the content of a document must be separate from
<!--42--> the style. Generic markup allows for multiple presentations of the
<!--43--> information.</para></topic>
<!--44--> <topic topicid="top3">
<!--45--> <title>Drawbacks of procedural markup</title>
<!--46--> <para>Industries involved in technical documentation increasingly
<!--47--> prefer generic over procedural markup schemes. When a company changes
<!--48--> software or hardware systems, enormous data translation tasks arise,
<!--49--> often resulting in errors.</para></topic></section>
<!--50--> <section shorttitle="What is SGML?">
<!--51--> <title>What <emph>is</emph> SGML in the grand scheme of the universe, an
<!--52--> <intro>
<!--53--> <para>SGML defines a strict markup scheme with a syntax for defining
<!--54--> document data elements and an overall framework for marking up
```

```
<!--55--> documents.</para>
<!--56--> <para>SGML can describe and create documents that are not dependent on
<!--57--> any hardware, software, formatter, or operating system. Since SGML docume
<!--58--> conform to an international standard, they are portable.</para></intro></
<!--59--> <section shorttitle="How does SGML work?">
<!--60--> <title>How is SGML and would you recommend it to your grandmother?</title
<!--61--> <intro>
<!--62--> <para>You can break a typical document into three layers: structure,
<!--63--> content, and style. SGML works by separating these three aspects and
<!--64--> deals mainly with the relationship between structure and content.</para><
<!--65--> <topic topicid="top4">
<!--66--> <title>Structure</title>
<!--67--> <para>At the heart of an SGML application is a file called the DTD, or
<!--68--> Document Type Definition. The DTD sets up the structure of a document,
<!--69--> much like a database schema describes the types of information it
<!--70--> handles.</para>
<!--71--> <para>A database schema also defines the relationships between the
<!--72--> various types of data. Similarly, a DTD specifies <emph>rules</emph>
<!--73--> to help ensure documents have a consistent, logical structure.</para></to
<!--74--> <topic topicid="top5">
<!--75--> <title>Content</title>
<!--76--> <para>Content is the information itself. The method for identifying
<!--77--> the information and its meaning within this framework is called
<!--78--> <emph>tagging</emph>. Tagging must
<!--79--> conform to the rules established in the DTD (see <xref xrefid="top4"/>).<
<!--80--> <graphic graphname="tagexamp"/></topic>
<!--81--> <topic topicid="top6">
<!--82--> <title>Style</title>
<!--83--> <para>SGML does not standardize style or other processing methods for
<!--84--> information stored in SGML.</para></topic></section></chapter>
<!--85--> <chapter>
<!--86--> <title>Resources</title>
<!--87--> <section>
<!--88--> <title>Conferences, tutorials, and training</title>
<!--89--> <intro>
<!--90--> <para>The Graphic Communications Association has been
<!--91--> instrumental in the development of SGML. GCA provides conferences,
<!--92--> tutorials, newsletters, and publication sales for both members and
<!--93--> non-members.</para>
<!--94--> <para security="c">Exiled members of the former Soviet Union's secret
<!--95--> police, the KGB, have infiltrated the upper ranks of the GCA and are
<!--96--> planning the Final Revolution as soon as DSSSL is completed.</para>
<!--97--> </intro>
<!--98--> </section>
<!--99--> </chapter>
<!--100--></report>
```

1.5.4 Queries and Results

1.5.4.1 Q1

Locate all paragraphs in the report (all "para" elements occurring anywhere within the "report" element).

Solution in XQuery:

```
<result>
```

```
  {
    //report//para
  }
</result>
```

Expected Result:

Elements whose start-tags are on lines 6, 11, 20, 27, 34, 39, 46, 53, 56, 62, 67, 71, 76, 83, 90, 94

1.5.4.2 Q2

Locate all paragraph elements in an introduction (all "para" elements directly contained within an "intro" element).

Solution in XQuery:

```
<result>
  {
    //intro/para
  }
</result>
```

Expected Result:

Elements whose start-tags are on lines 6, 11, 20, 27, 53, 56, 62, 90, 94

1.5.4.3 Q3

Locate all paragraphs in the introduction of a section that is in a chapter that has no introduction (all "para" elements directly contained within an "intro" element directly contained in a "section" element directly contained in a "chapter" element. The "chapter" element must not directly contain an "intro" element).

Solution in XQuery:

```
<result>
  {
         FOR $c IN //chapter
         WHERE empty($c/intro)
         RETURN $c/section/intro/para
  }
</result>
```

Expected Result:

Elements whose start-tags are on lines 90, 94

1.5.4.4 Q4

Locate the second paragraph in the third section in the second chapter (the second "para" element occurring in the third "section" element occurring in the second "chapter" element occurring in the

"report").

Solution in XQuery:

```
<result>
  {
    (((//chapter)[2]//section)[3]//para)[2]
  }
</result>
```

Expected Result:

Element whose start-tag is on line 67

1.5.4.5 Q5

Locate all classified paragraphs (all "para" elements whose "security" attribute has the value "c").

Solution in XQuery:

```
<result>
  {
    //para[@security = "c"]
  }
</result>
```

Expected Result:

Element whose start-tag is on line 94

1.5.4.6 Q6

List the short titles of all sections (the values of the "shorttitle" attributes of all "section" elements, expressing each short title as the value of a new element.)

Solution in XQuery:

```
<result>
  {
         FOR $s IN //section/@shorttitle
         RETURN <stitle>{ $s }</stitle>
  }
</result>
```

Expected Result:

Attribute values in start-tags on lines 23, 50, 59

1.5.4.7 Q7

Locate the initial letter of the initial paragraph of all introductions (the first character in the content [character content as well as element content] of the first "para" element contained in an "intro" element).

Solution in XQuery:

```
<result>
  {
         FOR $i IN //intro/para[1]
         RETURN
                  <first_letter>{ substring(string($i), 1, 1) }</first_letter>
  }
</result>
```

Expected Result:

Character after start-tag on lines 6, 20, 27, 53, 62, 90

1.5.4.8 Q8a

Locate all sections with a title that has "is SGML" in it (all "section" elements that contain a "title" element that has the consecutive characters "is SGML" in its content). The string can be interrupted by sub-elements.

Solution in XQuery:

```
<result>
  {
    //section[contains(string(.//title), "is SGML")]
  }
</result>
```

Expected Result:

Elements whose start-tags are on lines 51, 60

1.5.4.9 Q8b

Same as (Q8a), but the string cannot be interrupted by sub-elements.

Solution in XQuery:

```
<result>
  {
    //section[contains(.//title/text(), "is SGML")]
  }
</result>
```

Expected Result:

Element whose start-tag is on line 60

1.5.4.10 Q9

Locate all the topics referenced by a cross-reference anywhere in the report (all the "topic" elements whose "topicid" attribute value is the same as an "xrefid" attribute value of any "xref" element).

Solution in XQuery:

```
<result>
  {
         FOR $id IN document("input.xml")//xref/@xrefid
         RETURN //topic[@topicid = $id]
  }
</result>
```

Expected Result:

Element whose start-tag is on line 65

1.5.4.11 Q10

Locate the closest title preceding the cross-reference ("xref") element whose "xrefid" attribute is "top4" (the "title" element that would be touched last before this "xref" element when touching each element in document order).

Solution in XQuery:

```
<result>
  {
         LET $x := //xref[@xrefid = "top4"],
             $t := //title BEFORE $x
         RETURN $t[last()]
  }
</result>
```

Expected Result:

Given xref on line 79, element whose start-tag is on line 75

1.6 Use Case "TEXT": Full-text Search

1.6.1 Description

This use case is based on company profiles and a set of news documents which contain data for PR, mergers and acquisitions, etc. Given a company, the use case illustrates several different queries for searching text in news documents and different ways of providing query results by matching the information from the company profile and the content of the news items.

In this use case, searches for company names are to be interpreted as word-based searches. The

words in a company name may be in any case and may be separated by any kind of white space.

1.6.2 Document Type Definition (DTD)

This use case is based on an implicit (unnamed) input data set based on the following DTDs:

```
<!ELEMENT company (name, ticker_symbol?, description?,
          business_code, partners?, competitors?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT ticker_symbol (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT business_code (#PCDATA)>
<!ELEMENT partners (partner+)>
<!ELEMENT partner (#PCDATA)>
<!ELEMENT competitors (competitor+)>
<!ELEMENT competitor (#PCDATA)>

<!ELEMENT news (news_item*)>
<!ELEMENT news_item (title, content, date, author?, news_agent)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT content (par | figure)+ >
<!ELEMENT date (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT news_agent (#PCDATA)>
<!ELEMENT par (#PCDATA | quote | footnote)*>
<!ELEMENT quote (#PCDATA)>
<!ELEMENT footnote (#PCDATA)>
<!ELEMENT figure (title, image)>
<!ELEMENT image EMPTY>
<!ATTLIST image
    source  CDATA  #REQUIRED >
```

1.6.3 Sample Data

The queries in this use case are based on the following sample data.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<news>
<news_item>
   <title> Gorilla Corporation acquires YouNameItWeIntegrateIt.com </title>
   <content>
      <par>
            Today, Gorilla Corporation announced that it will purchase
          YouNameItWeIntegrateIt.com. The shares of YouNameItWeIntegrateIt.com
          dropped $3.00 as a result of this announcement.
      </par>

      <par>
            As a result of this acquisition, the CEO of YouNameItWeIntegrateIt.com
          Bill Smarts resigned. He did not announce what he will do next.
          Sources close to YouNameItWeIntegrateIt.com hint that Bill Smarts
          might be taking a position in Foobar Corporation.
      </par>

      <par>
            YouNameItWeIntegrateIt.com is a leading systems integrator that enables
          <quote>brick and mortar</quote> companies to have a presence on the web.
      </par>
```

```
   </content>
   <date>1-20-2000</date>
   <author>Mark Davis</author>
   <news_agent>News Online</news_agent>
</news_item>

<news_item>
   <title>Foobar Corporation releases its new line of Foo products today</title>
   <content>
      <par>
            Foobar Corporation releases the 20.9 version of its Foo products.
          The new version of Foo products solve known performance problems which
          existed in 20.8 line and increases the speed of Foo based products
          tenfold. It also allows wireless clients to be connected to the Foobar
          servers.
      </par>
      <par>
            The President of Foobar Corporation announced that they were proud to
          release 20.9 version of Foo products and they will upgrade existing
          customers <footnote>where service agreements exist</footnote>
          promptly. TheAppCompany Inc. immediately announced that it
          will release the new version of its products to utilize the 20.9
          architecture within the next three months.
      </par>
      <figure>
          <title>Presidents of Foobar Corporation and TheAppCompany Inc. Shake Hanc
          <image source="handshake.jpg"/>
      </figure>
   </content>
   <date>1-20-2000</date>
   <news_agent>Foobar Corporation</news_agent>
</news_item>

<news_item>
   <title>Foobar Corporation is suing Gorilla Corporation for patent infringement <
   <content>
      <par>
           In surprising developments today, Foobar Corporation announced that
         it is suing Gorilla Corporation for patent infringement. The patents
         that were mentioned as part of the lawsuit are considered to be the
         basis of Foobar Corporation's <quote>Wireless Foo</quote> line of
            products.
      </par>
      <par>
           The tension between Foobar and Gorilla Corporations has been increasing
         ever since the Gorilla Corporation acquired more than 40 engineers who
         have left Foobar Corporation, TheAppCompany Inc. and
         YouNameItWeIntegrateIt.com over the past 3 months. The engineers who
         have left the Foobar corporation and its partners were rumored to be
         working on the next generation of server products and applications
         which will directly compete with Foobar's Foo 20.9 servers. Most of the
         engineers have relocated to Hawaii where the Gorilla Corporation's
         server development is located.
      </par>
   </content>
   <date>1-20-2000</date>
   <news_agent>Reliable News Corporation</news_agent>
</news_item>
</news>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE company SYSTEM "company.dtd">
<company>
   <name>Foobar Corporation</name>
   <ticker_symbol>FOO</ticker_symbol>
   <description>Foobar Corporation is a maker of Foo(TM) and Foobar(TM)
        products and a leading software company with a 300 Billion dollar
        revenue in 1999. It is located in Alaska.
   </description>
   <business_code>Software</business_code>
   <partners>
        <partner>YouNameItWeIntegrateIt.com</partner>
        <partner>TheAppCompany Inc.</partner>
   </partners>
   <competitors>
        <competitor>Gorilla Corporation</competitor>
   </competitors>
</company>
```

1.6.4 Queries and Results

1.6.4.1 Q1

Find all news items where the name "Foobar Corporation" appears in the title.

Solution in XQuery:

```
//news_item/title[contains(./text(), "Foobar Corporation")]
```

Expected Results

```
<title>Foobar Corporation releases its new line of Foo products today</title>
<title>Foobar Corporation is suing Gorilla Corporation for patent infringement </ti
```

1.6.4.2 Q2

Find news items where the Foobar Corporation and one or more of its partners are mentioned in the same paragraph and/or title. List each news item by its title and date.

Solution in XQuery:

```
LET $foobar_partners := //company[name = "Foobar Corporation"]//partner
FOR $item IN //news_item
WHERE
  SOME $t IN $item//title SATISFIES
    (contains($t/text(), "Foobar Corporation")
    AND SOME $partner IN $foobar_partners SATISFIES
      contains($t/text(), $partner/text()))
  OR SOME $par IN $item//par SATISFIES
   (contains($par/text(), "Foobar Corporation")
```

```
     AND SOME $partner IN $foobar_partners SATISFIES
       contains($par/text(), $partner/text()))
RETURN
         <news_item>
                  { $item/title }
                  { $item/date }
         </news_item>
```

Expected Result:

```
<news_item>
         <title> Gorilla Corporation acquires YouNameItWeIntegrateIt.com </title>
         <date>1-20-2000</date>
</news_item>
<news_item>
         <title>Foobar Corporation releases its new line of Foo products today</tit
         <date>1-20-2000</date>
</news_item>
<news_item>
         <title>Foobar Corporation is suing Gorilla Corporation for patent infringe
         <date>1-20-2000</date>
</news_item>
```

1.6.4.3 Q3

Find titles of news items where Foobar Corporation and one or more of its partners are mentioned in the same sentence, but none of its competitors are mentioned in the news item. (The "." character designates the end of a sentence.)

Solution in XQuery:

```
LET $foobar_partners := //company[name = "Foobar Corporation"]//partner,
    $foobar_competitors := //company[name = "Foobar Corporation"]//competitor
FOR $item IN //news_item
WHERE SOME $partner IN $foobar_partners SATISFIES
  contains_in_same_sentence(string($item/content), "Foobar Corporation", $partner/t
      AND not(SOME $competitor IN $foobar_competitors SATISFIES
                           contains(string($item/content), $competitor/text()))
RETURN
         $item/title
```

Expected Result:

```
<title>Foobar Corporation releases its new line of Foo products today </title>
```

Ed. Note: contains_in_same_sentence() is a function created on the spot to solve this use case.

1.6.4.4 Q4

Find news items where a company and one of its partners is mentioned in the same news item and the news item is not authored by the company itself.

Solution in XQuery:

```
FOR $item IN //news_item,
    $c IN //company
LET $partners := $c//partner
WHERE contains(string($item), $c/name/text())
  AND SOME $p IN $partners SATISFIES
    contains(string($item), $p/text()) AND $item/news_agent != $c/name
RETURN
         $item
```

Expected Results: The expected results are the news item elements with the following titles:

- Gorilla Corporation acquires YouNameItWeIntegrateIt.com
- Foobar Corporation is suing Gorilla Corporation for patent infringement

1.6.4.5 Q5

For each news item that is relevant to the Gorilla Corporation, create an "item summary" element. The content of the item summary is the content of the title, date, and first paragraph of the news item, separated by periods. A news item is relevant if the name of the company is mentioned anywhere within the content of the news item.

Solution in XQuery:

```
FOR $item IN //news_item
WHERE contains(string($item/content), "Gorilla Corporation")
RETURN
         <item_summary>
                  { $item/title/text() }.
                  { $item/date/text() }.
                  { string(($item//par)[1]) }
         </item_summary>
```

Expected Result: (with whitespace reformatted for readability)

```
<item_summary>
   Gorilla Corporation acquires YouNameItWeIntegrateIt.com. 1-20-2000.
   Today, Gorilla Corporation announced that it will purchase
   YouNameItWeIntegrateIt.com. The shares of YouNameItWeIntegrateIt.com
   dropped $3.00 as a result of this announcement.
</item_summary>

<item_summary>
   Foobar Corporation is suing Gorilla Corporation for patent infringement.
```

```
   1-20-2000. In surprising developments today, Foobar Corporation announced
   that it is suing Gorilla Corporation for patent infringement. The patents
   that were mentioned as part of the lawsuit are considered to be the
   basis of Foobar Corporation's <quote>Wireless Foo</quote> line of products.
</item_summary>
```

1.6.4.6 Q6

Find news items where two company names and some form of the word "acquire" appear in the title or in the same sentence in one of the paragraphs. A company name is defined as the content of a <name>, <partner>, or <competitor> element within a <company> element.

Solution in XQuery:

```
LET $companies := distinct(//company/name/text()
       UNION //company//partner/text()
       UNION //company//competitor/text())
FOR $item IN //news_item,
    $item_title IN $item/title,
    $item_para IN $item//par,
    $c1 IN $companies,
    $c2 IN $companies
WHERE $c1 != $c2
  AND contains_stems_in_same_sentence($item_title/text(), $c1, $c2, "acquire")
   OR contains_stems_in_same_sentence($item_para/text(), $c1, $c2, "acquire")
RETURN
         distinct($item)
```

Ed. Note: contains_stems_in_same_sentence() is a function created on the spot to solve this use case.

Expected Results: The expected results are the news item elements with the following titles:

- Gorilla Corporation acquires YouNameItWeIntegrateIt.com
- Foobar Corporation is suing Gorilla Corporation for patent infringement

1.7 Use Case "NS" - Queries Using Namespaces

This use case performs a variety of queries on namespace-qualified names.

1.7.1 Description

This use case is based on a scenario in which a neutral mediator is acting with public auction servers on behalf of clients. The reason for a client to use this imaginary service may be anonymity, better insurance, or the possibility to cover more than one market at a time. The following aspects of namespaces are illustrated by this use case:

- Syntactic disambiguation when combining XML data from different sources
- Re-use of predefined modules, such as XLinks or XML Schema

- Support for global classification schemas, such as the Dublin Core

The sample data consists of two records. The schema used for this data uses W3C XML Schema's schema composition to create a schema from predefined, namespace separated modules, and uses XLink to express references. Each record describes a running auction. It embeds data specific to an auctioneer (e.g. the company's credit rating system) and a taxonomy specific to a particular good (jazz records) in a framework that contains data common to all auctions (e.g. start and end time), using namespaces to distinguish the three vocabularies.

Note that namespace prefixes must be resolved to their Namespace URIs before matching namespace qualified names. It is not sufficient to use the literal prefixes to denote namespaces. Furthermore, there are several possible ways to represent namespace declarations. Therefore, processing must be done on the namespace processed XML Information Set, not on the XML text representation.

1.7.2 Document Type Definition (DTD)

DTDs are not fully compatible with namespaces as they can not express the equality of nodes in the same namespace, but different namespace proxies. In a later version of this paper, an XML Schema should be added here.

This use case is based on an implicit (unnamed) input data set.

1.7.3 Sample Data

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<ma:AuctionWatchList
    xmlns:ma="http://www.example.com/AuctionWatch"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:anyzone="http://www.example.com/auctioneers#anyzone"
    xmlns:eachbay="http://www.example.com/auctioneers#eachbay"
    xmlns:yabadoo="http://www.example.com/auctioneers#yabadoo"
>

<!-- ____________________________________________________________________

<ma:Auction anyzone:ID="0321K372910">

    <ma:AuctionHomepage
        xlink:type="simple"
        xlink:href="http://www.example.com/item/0321K372910"
    />

    <ma:Schedule>
        <ma:Open    xmlns:dt="http://www.w3.org/1999/XMLSchema-datatypes"
                dt:type="timeInstant">2000-03-21:07:41:34-05:00</ma:Open>
        <ma:Close    xmlns:dt="http://www.w3.org/1999/XMLSchema-datatypes"
                dt:type="timeInstant">2000-03-23:07:41:34-05:00</ma:Close>
    </ma:Schedule>

    <ma:Price>
        <ma:Start ma:currency="USD">3.00</ma:Start>
        <ma:Current ma:currency="USD">10.00</ma:Current>
        <ma:Number_of_Bids>5</ma:Number_of_Bids>
    </ma:Price>
```

```
        <ma:Trading_Partners>
            <ma:High_Bidder>
                <eachbay:ID>RecordsRUs</eachbay:ID>
                <eachbay:PositiveComments>231</eachbay:PositiveComments>
                <eachbay:NeutralComments>2</eachbay:NeutralComments>
                <eachbay:NegativeComments>5</eachbay:NegativeComments>
                <ma:MemberInfoPage
                    xlink:type="simple"
                    xlink:href="http://auction.eachbay.com/members?get=RecordsRUs"
                    xlink:role="ma:MemberInfoPage"
                />
            </ma:High_Bidder>
            <ma:Seller>
                <anyzone:ID>VintageRecordFreak</anyzone:ID>
                <anyzone:Member_Since>October 1999</anyzone:Member_Since>
                <anyzone:Rating>5</anyzone:Rating>
                <ma:MemberInfoPage
                    xlink:type="simple"
                    xlink:href="http://auction.anyzone.com/members/VintageRecordFreak"
                    xlink:role="ma:MemberInfoPage"
                />
            </ma:Seller>
        </ma:Trading_Partners>

        <ma:Details>
            <record xmlns="http://www.example.org/music/records">
                <artist>Miles Davis</artist>
                <title>In a Silent Way</title>
                <recorded>1969</recorded>
                <label>Columbia Records</label>
                <remark>
                    With Miles Davis (trumpet), Herbie Hancock (Electric Piano),
                    Chick Corea (Electric Piano), Wayne Shorter (Tenor Sax), Josef Zawi
                    (Electric Piano &amp;amp; Organ), John McLaughlin (Guitar), and
                    Tony Williams ( Drums).    The liner notes were written by Frank Gl
                    and the record is in fine condition.
                </remark>
            </record>
        </ma:Details>

    </ma:Auction>

    <!-- ____________________________________________________________________


    <ma:Auction yabadoo:ID="13143816">

        <ma:AuctionHomepage
            xlink:type="simple"
            xlink:href="http://auctions.yabadoo.com/auction/13143816"
        />

        <ma:Schedule>
            <ma:Open    xmlns:dt="http://www.w3.org/1999/XMLSchema-datatypes"
                    dt:type="timeInstant">2000-03-19:17:03:00-04:00</ma:Open>
            <ma:Close    xmlns:dt="http://www.w3.org/1999/XMLSchema-datatypes"
                    dt:type="timeInstant">2000-03-29:17:03:00-04:00</ma:Close>
        </ma:Schedule>
```

```
        <ma:Price>
            <ma:Start ma:currency="USD">3.00</ma:Start>
            <ma:Current ma:currency="USD">3.00</ma:Current>
            <ma:Number_of_Bids>0</ma:Number_of_Bids>
        </ma:Price>

        <ma:Trading_Partners>
            <ma:High_Bidder>
                <eachbay:ID>VintageRecordFreak</eachbay:ID>
                <eachbay:PositiveComments>232</eachbay:PositiveComments>
                <eachbay:NeutralComments>0</eachbay:NeutralComments>
                <eachbay:NegativeComments>0</eachbay:NegativeComments>
                <ma:MemberInfoPage
                    xlink:type="simple"
                    xlink:href="http://auction.eachbay.com/showRating/user=VintageRecor
                    xlink:role="ma:MemberInfoPage"
                />
            </ma:High_Bidder>
            <ma:Seller xmlns:seller="http://www.example.com/auctioneers#eachbay">
                <seller:ID>StarsOn45</seller:ID>
                <seller:PositiveComments>80</seller:PositiveComments>
                <seller:NeutralComments>1</seller:NeutralComments>
                <seller:NegativeComments>2</seller:NegativeComments>
                <ma:MemberInfoPage
                    xlink:type="simple"
                    xlink:href="http://auction.eachbay.com/showRating/user=StarsOn45"
                    xlink:role="ma:MemberInfoPage"
                />
            </ma:Seller>
        </ma:Trading_Partners>

        <ma:Details>
            <record xmlns="http://www.example.org/music/records">
                <artist>Wynton Marsalis</artist>
                <title>Think of One ...</title>
                <recorded>1983</recorded>
                <label>Columbia Records</label>
                <remark xml:lang="en">
                    Columbia Records 12" 33-1/3 rpm LP, #FC-38641, Stereo. The record i
                    still clean and shiny and looks unplayed (looks like NM condition).
                    The cover has very light surface and edge wear.
                </remark>
                <remark xml:lang="de">
                    Columbia Records 12" 33-1/3 rpm LP, #FC-38641, Stereo. Die Platte
                    ist noch immer sauber und gl? end und sieht ungespielt aus
                    (NM Zustand). Das Cover hat leichte Abnutzungen an Oberfl? e und Ec
                </remark>

            </record>
        </ma:Details>

    </ma:Auction>

    </ma:AuctionWatchList>
```

1.7.4 Queries and Results

1.7.4.1 Q1

List all unique namespaces used in the sample data.

Solution in XQuery:

```
<Q1>
  {
         FOR $n IN distinct(namespace_uri(//*))
         RETURN
                  $n + newline()
  }
</Q1>
```

Expected Result:

```
<Q1>
  http://www.example.com/AuctionWatch
  http://www.w3.org/1999/xlink
  http://www.example.com/auctioneers#anyzone
  http://www.example.com/auctioneers#eachbay
  http://www.example.com/auctioneers#yabadoo
  http://www.w3.org/1999/XMLSchema-datatypes
  http://www.example.org/music/records
</Q1>
```

1.7.4.2 Q2

Select the title of each record that is for sale.

Solution in XQuery:

```
NAMESPACE music = "http://www.example.org/music/records"

<Q2>
  {
    //music:title
  }
</Q2>
```

Expected Result:

```
<Q2 xmlns:music="http://www.example.org/music/records">
  <music:title>In a Silent Way</music:title>
  <music:title>Think of One ...</music:title>
</Q2>
```

1.7.4.3 Q3

Select all elements using datatypes from "XML Schema: Part 2" datatypes.

Solution in XQuery:

```
NAMESPACE dt = "http://www.w3.org/1999/XMLSchema-datatypes"

<Q3>
  {
    //*[@dt:*]
  }
</Q3>
```

(Finds all nodes that have an attribute whose namespace matches the one specified.)

Expected Result:

```
<Q3 xmlns:dt="http://www.w3.org/1999/XMLSchema-datatypes"
        xmlns:ma="http://www.example.com/AuctionWatch"
>
  <ma:Open dt:type="timeInstant">2000-03-21:07:41:34-05:00</ma:Open>
  <ma:Close dt:type="timeInstant">2000-03-23:07:41:34-05:00</ma:Close>
  <ma:Open dt:type="timeInstant">2000-03-19:17:03:00-04:00</ma:Open>
  <ma:Close dt:type="timeInstant">2000-03-29:17:03:00-04:00</ma:Close>
</Q3>
```

1.7.4.4 Q4

List the target URI's of all XLinks in the document.

Solution in XQuery:

```
NAMESPACE xlink = "http://www.w3.org/1999/xlink"

<Q4>
  {
        FOR $hr IN //@xlink:href
        RETURN $hr + newline()
  }
</Q4>
```

(Binds $hr to attribute nodes named href. Concatenating the attribute node to a string extracts its value.)

Expected Result:

```
<Q4>
  http://auction.anyzone.com/item/0321K372910
  http://auction.eachbay.com/members?get=RecordsRUs
  http://auction.anyzone.com/members/VintageRecordFreak
  http://auctions.yabadoo.com/auction/13143816
  http://auction.eachbay.com/showRating/user=VintageRecordFreak
```

```
  http://auction.eachbay.com/showRating/user=StarsOn45
</Q4>
```

1.7.4.5 Q5

Select all records that have a remark in German.

Solution in XQuery:

```
NAMESPACE music = "http://www.example.org/music/records"

<Q5>
  {
     //music:record[music:remark/xml:lang = "de"]
  }
</Q5>
```

Expected Result:

```
<Q5 xmlns:music="http://www.example.org/music/records">
<music:record>
    <music:artist>Wynton Marsalis</music:artist>
    <music:title>Think of One ...</music:title>
    <music:recorded>1983</music:recorded>
    <music:label>Columbia Records</music:label>
    <music:remark xml:lang="en">
        Columbia Records 12" 33-1/3 rpm LP, #FC-38641, Stereo. The record is
        still clean and shiny and looks unplayed (looks like NM condition).
        The cover has very light surface and edge wear.
    </music:remark>
    <music:remark xml:lang="de">
        Columbia Records 12" 33-1/3 rpm LP, #FC-38641, Stereo. Die Platte
        ist noch immer sauber und gl? end und sieht ungespielt aus
        (NM Zustand). Das Cover hat leichte Abnutzungen an Oberfl? e und Ecken.
    </music:remark>
</music:record>
</Q5>
```

1.7.4.6 Q6

Select the closing time elements of all AnyZone auctions currently monitored.

Solution in XQuery:

```
NAMESPACE ma = "http://www.example.com/AuctionWatch"
NAMESPACE anyzone = "http://www.example.com/auctioneers#anyzone"

<Q6>
  {
    //ma:Auction[@anyzone:ID]/ma:Schedule/ma:Close
  }
</Q6>
```

Expected Result:

```
<Q6 xmlns:dt="http://www.w3.org/1999/XMLSchema-datatypes"
xmlns:ma="http://www.example.com/AuctionWatch">
  <ma:Close dt:type="timeInstant">2000-03-23:07:41:34-05:00</ma:Close>
</Q6>
```

1.7.4.7 Q7

Select the homepage of all auctions where both seller and high bidder are registered at the same auctioneer.

Solution in XQuery:

```
NAMESPACE ma = "http://www.example.com/AuctionWatch"

<Q7>
  {
         FOR $a IN //ma:Auction
         LET $seller_id := $a/ma:Trading_Partners/ma:Seller/*:ID,
             $buyer_id := $a/ma:Trading_Partners/ma:High_Bidder/*:ID
         WHERE namespace_uri($seller_id) = namespace_uri($buyer_id)
         RETURN
                  $a/ma:AuctionHomepage
  }
</Q7>
```

Expected Result:

```
<Q7 xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:ma="http://www.example.com/AuctionWatch" >
  <ma:AuctionHomepage xlink:type="simple"
                      xlink:href="http://auctions.yabadoo.com/auction/13143816" />
</Q7>
```

1.7.4.8 Q8

Select all traders (either seller or high bidder) without negative comments

Solution in XQuery:

```
NAMESPACE ma = "http://www.example.com/AuctionWatch"

<Q8>
  {
         (FOR $s IN //ma:Trading_Partners/ma:Seller
          WHERE $s/*:NegativeComments = 0
          RETURN $s)
  UNION
```

```
          (FOR $b IN //ma:Trading_Partners/ma:High_Bidder
           WHERE $b/*:NegativeComments = 0
           RETURN $b)
  }
</Q8>
```

Expected Result:

```
<Q8 xmlns:ma="http://www.example.com/AuctionWatch"
        xmlns:eachbay="http://www.example.com/auctioneers#eachbay"
        xmlns:xlink="http://www.w3.org/1999/xlink" >
<ma:High_Bidder>
  <eachbay:ID>VintageRecordFreak</eachbay:ID>
  <eachbay:PositiveComments>232</eachbay:PositiveComments>
  <eachbay:NeutralComments>0</eachbay:NeutralComments>
  <eachbay:NegativeComments>0</eachbay:NegativeComments>
  <ma:MemberInfoPage xlink:type="simple"
                     xlink:href="http://auction.eachbay.com/showRating/user=Vintage
                     xlink:role="ma:MemberInfoPage" />
</ma:High_Bidder>
</Q8>
```

1.8 Use Case "PARTS" - Recursive Parts Explosion

This use case illustrates how a recursive query might be used to construct a hierarchic document of arbitrary depth from flat structures stored in a database.

1.8.1 Description

This use case is based on a "parts explosion" database that contains information about how parts are used in other parts.

The input to the use case is a "flat" document in which each different part is represented by a <part> element with partid and name attributes. Each part may or may not be part of a larger part; if so, the partid of the larger part is contained in a partof attribute. This input document might be derived from a relational database in which each part is represented by a row of a table with partid as primary key and partof as a foreign key referencing partid.

The challenge of this use case is to write a query that converts the "flat" representation of the parts explosion, based on foreign keys, into a hierarchic representation in which part containment is represented by the structure of the document.

1.8.2 Document Type Definitions (DTD)

The input data set uses the following DTD:

```
<!DOCTYPE partlist [
    <!ELEMENT partlist (part*)>
    <!ELEMENT part EMPTY>
    <!ATTLIST part
          partid  CDATA  #REQUIRED
          partof  CDATA  #IMPLIED
          name    CDATA  #REQUIRED>
]>
```

Although the `partid` and `partof` attributes could have been of type ID and IDREF, respectively, in this schema they are treated as character data, possibly materialized in a straightforward way from a relational database. Each `partof` attribute matches exactly one `partid`. Parts having no `partof` attribute are not contained in any other part.

The output data conforms to the following DTD:

```
<!DOCTYPE parttree [
    <!ELEMENT parttree (part*)>
    <!ELEMENT part (part*)>
    <!ATTLIST part
          partid  CDATA  #REQUIRED
          name    CDATA  #REQUIRED>
]>
```

1.8.3 Sample Data

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<partlist>
  <part partid="0" name="car"/>
  <part partid="1" partof="0" name="engine"/>
  <part partid="2" partof="0" name="door"/>
  <part partid="3" partof="1" name="piston"/>
  <part partid="4" partof="2" name="window"/>
  <part partid="5" partof="2" name="lock"/>
  <part partid="10" name="skateboard"/>
  <part partid="11" partof="10" name="board"/>
  <part partid="12" partof="10" name="wheel"/>
  <part partid="20" name="canoe"/>
</partlist>
```

1.8.4 Queries and Results

1.8.4.1 Q1

Convert the sample document from "partlist" format to "parttree" format (see DTD section for definitions). In the result document, part containment is represented by containment of one <part> element inside another. Each part that is not part of any other part should appear as a separate top-level element in the output document.

Solution in XQuery:

```
DEFINE FUNCTION one_level (ELEMENT $p) RETURNS ELEMENT
{
         <part partid={ $p/@partid }
               name={ $p/@name } >
                  {
                           FOR $s IN document("data/parts-data.xml")//part
                           WHERE $s/@partof = $p/@partid
                           RETURN one_level($s)
                  }
         </part>
}
```

```
<parttree>
  {
        FOR $p IN document("data/parts-data.xml")//part[empty(@partof)]
        RETURN one_level($p)
  }
</parttree>
```

Expected Result:

```
<parttree>
        <part partid="0" name="car">
                <part partid="1" name="engine">
                        <part partid="3" name="piston"/>
                </part>
                <part partid="2" name="door">
                        <part partid="4" name="window"/>
                        <part partid="5" name="lock"/>
                </part>
        </part>
        <part partid="10" name="skateboard">
                <part partid="11" name="board"/>
                <part partid="12" name="wheel"/>
        </part>
        <part partid="20" name="canoe"/>
</parttree>
```

1.9 Use Case "REF" - Queries based on References

1.9.1 Description

References are an important aspect of XML. This use case describes a database in which references play a significant role, and contains several representative queries that exploit these references.

Suppose that the file "census.xml" contains an element for each person recorded in a recent census. For each person element, the person's name, job, and spouse (if any) are recorded as attributes. The "spouse" attribute is an IDREF-type attribute that matches the ID-type "name" attribute of the spouse element.

The parent-child relationship among persons is recorded by containment in the element hierarchy. In other words, the element that represents a child is contained within the element that represents the child's father or mother. Due to deaths, divorces, and remarriages, a child might be recorded under either its father or its mother (but not both). For the purposes of this exercise, the term "children of X" includes "children of the spouse of X." For example, if Joe and Martha are spouses, and Joe's element contains an element Sam, and Martha's element contains an element Dave, then Joe's children are considered to be Sam and Dave, and Martha's children are also considered to be Sam and Dave. Each person in the census has zero, one, or two parents.

1.9.2 Document Type Definitions (DTD)

This use case is based on an input document named "census.xml", with the following DTD:

```
<!DOCTYPE census [
  <!ELEMENT census (person*)>
  <!ELEMENT person (person*)>
```

```
  <!ATTLIST person
        name    ID      #REQUIRED
        spouse  IDREF   #IMPLIED
        job     CDATA   #IMPLIED >
]>
```

1.9.3 Sample Data

The following census data describes two friendly families that have several intermarriages.

```
<census>
  <person name="Bill" job="Teacher">
    <person name="Joe" job="Painter" spouse="Martha">
      <person name="Sam" job="Nurse">
        <person name="Fred" job="Senator" spouse="Jane">
        </person>
      </person>
      <person name="Karen" job="Doctor" spouse="Steve">
      </person>
    </person>
    <person name="Mary" job="Pilot">
      <person name="Susan" job="Pilot" spouse="Dave">
      </person>
    </person>
  </person>
  <person name="Frank" job="Writer">
    <person name="Martha" job="Programmer" spouse="Joe">
      <person name="Dave" job="Athlete" spouse="Susan">
      </person>
    </person>
    <person name="John" job="Artist">
      <person name="Helen" job="Athlete">
      </person>
      <person name="Steve" job="Accountant" spouse="Karen">
        <person name="Jane" job="Doctor" spouse="Fred">
        </person>
      </person>
    </person>
  </person>
</census>
```

1.9.4 Queries and Results

1.9.4.1 Q1

Find Martha's spouse.

Solution in XQuery:

```
<result>
  {
        FOR $m IN document("census.xml")//person[name = "Martha"]
        RETURN shallow($m/@spouse->person)
  }
</result>
```

Expected Result:

```
<person name="Joe" job="Painter" spouse="Martha" />
```

1.9.4.2 Q2

Find Joe's children.

Solution in XQuery:

```
DEFINE FUNCTION children (ELEMENT $p)
{
         shallow($p/person) UNION shallow($p/@spouse->person/person)
}

<result>
  {
         FOR $j IN document("census.xml")//person[name = "Joe"]
         RETURN children($j)
  }
</result>
```

Expected Result:

```
<result>
  <person name="Sam" job="Nurse" />
  <person name="Karen" job="Doctor" spouse="Steve" />
  <person name="Dave" job="Athlete" spouse="Susan" />
</result>
```

1.9.4.3 Q3

Find parents of athletes.

Solution in XQuery:

```
<result>
  {
         FOR $p IN document("census.xml")//person,
             $s IN $p/@spouse->person
         WHERE $p/person/job = "Athlete" OR $s/person/job = "Athlete"
         RETURN shallow($p)
  }
</result>
```

Expected Result:

```
<result>
  <person name="Joe" job="Painter" spouse="Martha" />
  <person name="Martha" job="Programmer" spouse="Joe" />
  <person name="John" job="Artist" />
</result>
```

1.9.4.4 Q4

Find people who have the same job as one of their parents.

Solution in XQuery:

```
<result>
  {
         FOR $p IN document("census.xml")//person,
             $c IN $p/person
         WHERE $p/job = $c/job OR $p/@spouse->person/job = $c/job
         RETURN shallow($c)
  }
</result>
```

Expected Result:

```
<result>
  <person name="Susan" job="Pilot" spouse="Dave" />
  <person name="Jane" job="Doctor" spouse="Fred" />
</result>
```

1.9.4.5 Q5

List names of parents and children who have the same job, and their jobs.

Solution in XQuery:

```
<result>
         {
                  FOR $p IN document("census.xml")//person,
                      $c IN $p/person[job = $p/job]
                  RETURN
                           <match parent={ $p/name } child={ $c/name } job={ $c/jo
         }
         {
                  FOR $p IN document("census.xml")//person,
                      $c IN $p/@spouse->person/person[job = $p/job]
                  RETURN
                           <match parent={ $p/name } child={ $c/name } job={ $c/jo
         }
</result>
```

Expected Result:

```
<result>
  <match parent="Mary" child="Susan" job="Pilot" />
  <match parent="Karen" child="Jane" job="Doctor" />
</result>
```

1.9.4.6 Q6

Find Bill's grandchildren.

Solution in XQuery:

```
<result>
  {
         FOR $b IN document("census.xml")//person[name = "Bill"],
             $c IN $b/person | $b/@spouse->person/person,
             $g IN $c/person | $c/@spouse->person/person
         RETURN shallow($g)
  }
</result>
```

Expected Result:

```
<result>
  <person name="Sam" job="Nurse" />
  <person name="Karen" job="Doctor" spouse="Steve" />
  <person name="Susan" job="Pilot" spouse="Dave" />
  <person name="Dave" job="Athlete" spouse="Susan" />
</result>
```

1.9.4.7 Q7

List name-pairs of grandparents and grandchildren.

Solution in XQuery:

```
<result>
  {
         FOR $b IN document("census.xml")//person,
             $c IN $b/person | $b/@spouse->person/person,
             $g IN $c/person | $c/@spouse->person/person
         RETURN
                  <grandparent name={ $b/name } grandchild={ $g/name }/>
  }
</result>
```

Expected Result:

```
<result>
```

```
  <grandparent name="Bill" grandchild="Sam" />
  <grandparent name="Bill" grandchild="Karen" />
  <grandparent name="Bill" grandchild="Susan" />
  <grandparent name="Bill" grandchild="Dave" />
  <grandparent name="Frank" grandchild="Dave" />
  <grandparent name="Frank" grandchild="Helen" />
  <grandparent name="Frank" grandchild="Steve" />
  <grandparent name="Frank" grandchild="Sam" />
  <grandparent name="Frank" grandchild="Karen" />
  <grandparent name="Joe" grandchild="Fred" />
  <grandparent name="Joe" grandchild="Jane" />
  <grandparent name="Martha" grandchild="Fred" />
  <grandparent name="Martha" grandchild="Jane" />
  <grandparent name="John" grandchild="Jane" />
</result>
```

1.9.4.8 Q8

Find Dave's parents-in-law (parents of his spouse, if any).

Solution in XQuery:

```
<result>
  {
         FOR $s IN document("census.xml")//person[name = "Dave"]/@spouse->person,
             $sp IN $s/.. | $s/../@spouse->person
         RETURN
                  shallow($sp)
  }
</result>
```

Expected Result:

```
<result>
  <person name="Mary" job="Pilot" />
</result>
```

1.9.4.9 Q9

Find people with no children.

Solution in XQuery:

```
<result>
  {
         FOR $p IN document("census.xml")//person
         WHERE empty(children($p))
         RETURN shallow($p)
  }
</result>
```

(See Q2 for definition of children function.)

Expected Result:

```
<result>
  <person name="Fred" job="Senator" spouse="Jane" />
  <person name="Susan" job="Pilot" spouse="Dave" />
  <person name="Dave" job="Athlete" spouse="Susan" />
  <person name="Helen" job="Athlete" />
  <person name="Jane" job="Doctor" spouse="Fred" />
</result>
```

1.9.4.10 Q10

Find single parents (people with children but no spouse)

Solution in XQuery:

```
<result>
  {
         FOR $p IN document("census.xml")//person[person]
         WHERE empty($p/@spouse->person)
         RETURN shallow($p)
  }
</result>
```

Expected Result:

```
<result>
  <person name="Bill" job="Teacher" />
  <person name="Sam" job="Nurse" />
  <person name="Mary" job="Pilot" />
  <person name="Frank" job="Writer" />
  <person name="John" job="Artist" />
</result>
```

1.9.4.11 Q11

List the names of all Joe's descendants. Show each descendant as an element with the descendant's name as content and his or her marital status and number of children as attributes. Sort the descendants in descending order by number of children, and secondarily in alphabetical order by name.

Solution in XQuery:

```
DEFINE FUNCTION descrip (ELEMENT $e) RETURNS ELEMENT
{
         LET $kids := $e/* UNION $e/@spouse->person/*
         LET $mstatus :=  IF ($e[@spouse]) THEN "Yes" ELSE "No"
         RETURN
```

```
                    <person married={ $mstatus } nkids={ count($kids) }>{ $e/@name/t
}

DEFINE FUNCTION descendants (ELEMENT $e)
{
         IF (empty($e/* UNION $e/@spouse->person/*))
         THEN $e
         ELSE $e UNION descendants($e/* UNION $e/@spouse->person/*)
}

descrip(descendants(//person[name = "Joe"])) SORTBY(@nkids DESCENDING, .)
```

Expected Result:

```
<result>
  <descendant married="Yes" kids="1">Karen</descendant>
  <descendant married="No" kids="1">Sam</descendant>
  <descendant married="Yes" kids="0">Dave</descendant>
  <descendant married="Yes" kids="0">Fred</descendant>
  <descendant married="Yes" kids="0">Jane</descendant>
</result>
```

1.10 Use Case "FNPARM" - Functions and Parameters

This use case explores some of the ways in which functions can be defined and invoked. It is based on examples taken from XML Schema Part 0: Primer , and relies in some cases on type and element definitions that are found in that document (referred to hereafter as the "Schema Primer".) Examples have been taken from the Schema Primer because the XML Query Working Group is committed to supporting queries based on the types and elements that can be defined using XML Schema. Only relevant parts of the type and element definitions are repeated in this use case--additional details can be found in the Schema Primer.

In general, this use case uses unqualified names for schema objects (for example, `complexType` and `integer` rather than `xsd:complexType` and `xsd:integer`). This corresponds to a strategy of using the schema namespace as the default namespace. Obviously, other strategies exist in which namespace prefixes would be used with names of schema objects. Since this use case is not primarily about namespaces, an effort has been made to keep namespace prefixes to a minimum.

This use case consists of several "subcases," each of which introduces a schema fragment and one or more functions that operate on data that conforms to the schema fragment. Each subcase explores a particular aspect of function definition and invocation.

1.10.1 Subcase 1: Type Names vs. Element Names

Section 2.2 of the Schema Primer defines a complex type named `USAddress` and elements named `shipTo` and `billTo`, of type `USAddress`, that are used in purchase orders. Here are the partial definitions (see Schema Primer for details):

```
<complexType name="USAddress"> . . .

<element name="shipTo" type="po:USAddress"/>

<element name="billTo" type="po:USAddress"/>
```

This subcase defines a function that takes a parameter of type `po:USAddress` and returns an integer:

```
FUNCTION timezone(TYPE po:USAddress $a) RETURNS integer
   { . . . }
```

The `timezone` function can be invoked on any expression whose static type is `USAddress`. The following expression computes the difference between the timezones of the `shipTo` address and the `billTo` address of the purchase order bound to variable `$po1`:

```
timezone($po1/shipTo) - timezone($po1/billTo)
```

(Note: the static type of each argument is `USAddress`)

1.10.2 Subcase 2: Types Derived by Extension

Section 4.1 of the Schema Primer defines a complex type named `Address` and two derived types named `USAddress` and `UKAddress`, used in international purchase orders. Here are the partial definitions (see Schema Primer for details):

```
<complexType name="Address"> . . .

<complexType name="USAddress">
   . . . <extension base="ipo:Address"> . . .

<complexType name="UKAddress">
   . . . <extension base="ipo:Address"> . . .
```

This subcase defines a function that takes a parameter of type `Address` and returns a string:

```
FUNCTION streetname(TYPE ipo:Address $a) RETURNS string
   { . . . }
```

Suppose that a product can have two warehouses, one in the USA and one in the UK, represented by subelements with the following declarations:

```
<element name="USAWarehouse" type="ipo:USAddress"/>

<element name="UKWarehouse" type="ipo:UKAddress"/>
```

The following expressions use the `streetname` function to find the streetnames of the two warehouses used by the product bound to `$product1`:

```
streetname($product1/USAWarehouse)

streetname($product1/UKWarehouse)
```

1.10.3 Subcase 3: Types Derived by Restriction

Section 4.4 of the Schema Primer defines a complex type named `Items` and a derived type named `confirmedItems`. Here are the partial definitions (see Schema Primer for details):

```
<complexType name="Items">
   <sequence>
      <element name="item" minOccurs="0" maxOccurs="unbounded">
```

```
. . .

<complexType name="ConfirmedItems">
   <complexContent>
      <restriction base="ipo:Items">
         <sequence>
            <element name="item" minOccurs="1" maxOccurs="unbounded">
. . .
```

Motivation for this subcase comes from the following statements in Section 4.4 of the Schema Primer: "An application prepared for the values of the base type would not be surprised by the values of the restricted type." ... "All `ConfirmedItems`-type elements will also be acceptable as `Items`-type elements."

This subcase defines a function that takes a parameter of type `Items` and returns a decimal number:

```
FUNCTION totalPrice(TYPE ipo:Items $x) RETURNS decimal
   { sum($x/item/price) }
```

Suppose that an `order` element can contain subelements named `estimate`, of type `Items`, and `final`, of type `confirmedItems`. The following expressions invoke the `totalPrice` function on these subelements of the order bound to variable `$order1`:

```
totalPrice($order1/estimate)
```

(Note: static type of argument is `Items`)

```
totalPrice($order1/final)
```

(Note: static type of argument is `ConfirmedItems`)

1.10.4 Subcase 4: Simulated Polymorphism

Most modern computer languages support some form of polymorphism, in which a function call may result in different behaviors depending on the dynamic type of the argument. Polymorphism is often supported by function overloading, in which several functions are defined that have the same name but different signatures (parameter types). However, function overloading will not be supported in XML Query Version 1. Therefore a different mechanism for simulating polymorphism would be useful. This subcase explores one such mechanism.

The base type `Address` with derived types `USAddress` and `UKAddress`, defined in Section 4.1 of the Schema Primer, are repeated here (modified slightly, and showing only the parts relevant for this example):

```
<complexType name="Address"> . . .

<complexType name="USAddress">
   . . . <extension base="ipo:Address"> . . .
      . . . <element name="zip" type="string"/> . . .

<complexType name="UKAddress">
   . . . <extension base="ipo:Address"> . . .
      . . . <element name="postcode" type="string"/> . . .
```

This subcase defines a function whose formal parameter is of type `ipo:Address`. By the principle of

subtype substitution, the function can be invoked with an argument of dynamic type `po:USAddress` or `po:UKAddress`. The function branches on the dynamic type of its argument.

```
FUNCTION code(TYPE ipo:Address $a) RETURNS string
   {
   TYPESWITCH ($a)
      CASE ipo:USAddress RETURN (TREAT $a AS ipo:USAddress)/zip/data()
      CASE ipo:UKAddress RETURN (TREAT $a AS ipo:UKAddress)/postcode/data()
      ELSE RETURN "none"
   }
```

The behavior of the function call `code($addr)` depends on the dynamic type of the argument. If `$addr` is of type `ipo:USAddress`, the function will return the string content of a `zip` element. If `$addr` is of type `ipo:UKAddress`, the function will return the string content of a `postcode` element. Otherwise, the function will return the string "none".

1.10.5 Subcase 5: Substitution Groups

Section 4.6 of the Schema Primer defines an element named `comment`, and two other elements in the same substitution group named `shipComment` and `customerComment`. Here are the partial definitions (see Schema Primer for details):

```
<element name="comment" type="string" />

<element name="shipComment" type="string"
   substitutionGroup="ipo:comment" />
. . .

<element name="customerComment" type="string"
   subtitutionGroup="ipo:comment" />
. . .
```

This use case defines a function called `complaint` that determines whether a comment is a complaint, returning a boolean value:

```
FUNCTION complaint(ELEMENT ipo:comment $c) RETURNS boolean
   { . . . }
```

The following expressions invoke the `complaint` function on specific comments that are found in the purchase order bound to variable `$po1`:

```
complaint($po1/shipComment[1])

complaint($po1/customerComment[1])
```

1.10.6 Subcase 6: Collections

This subcase uses an element named `mouse` whose type is `Mouse`, as defined by the following schema fragment:

```
<element name="mouse" type="bio:Mouse"/>
```

The use case defines two functions: a `length` function, which operates on one mouse, and an `avgWeight` function, which operates on a collection of mice. In the function signature we use the keyword `LIST` to denote the collection:

```
FUNCTION complaint(ELEMENT ipo:comment $c) RETURNS boolean
   { . . . }
```

The following expressions invoke the `complaint` function on specific comments that are found in the purchase order bound to variable `$po1`:

```
FUNCTION length(ELEMENT bio:mouse $m) RETURNS float
   { $m/body + $m/tail }

FUNCTION avgWeight(LIST(ELEMENT bio:mouse) $m) RETURNS float
   { avg($m/weight) }
```

The following function invocations use variable `$one_mouse`, which is bound to one mouse, and expression `$zoo//mouse`, which evaluates to a collection of zero or more mice. Note the number of times the body of the function is executed in each case:

```
length($one_mouse)
     (body is executed once)

length($zoo//mouse)
     (body is executed multiple times, once for each mouse)

avgWeight($one_mouse)
     (body is executed once)

avgWeight($zoo//mouse)
     (body is executed once)
```

Note that the keyword `LIST` in the function signature determines how many times the body of the function will be executed when the function is invoked on a collection. Note also that a single element is treated as a list of length one when passed to a function that expects a list.

1.10.7 Subcase 7: Any Element

Occasionally it is useful to define a function that can be invoked on any element, regardless of its name or type. XML Schema provides a way to define a generic type that matches any element, as described in Section 5.5 of the Schema Primer and in Section 4.3.7 of Schema Part 1:

```
<complexType name="anyElement">
   <any processContents="lax"/>
</complexType>
```

In this subcase, we assume that this type definition, and other useful generic type definitions, are included in a standard namespace called `qt`. The following function makes use of this type definition to find all the red children of any element. This function also makes use of the LIST notation defined in subcase 6.

Note the use of the keyword MATCH in the function signature. We have now seen three keywords used in function signatures: ELEMENT denotes that the formal parameter is an element with a given name; TYPE denotes that the formal parameter is an element that is declared to have a given type; and MATCH denotes that the formal parameter is an element whose content matches the content model of the given type, regardless of its name.

```
FUNCTION redChildren(MATCH qt:anyElement $e)
   RETURNS LIST(MATCH qt:anyElement)
```

```
{ $e/*[color = "Red"] }
```

The function `redChildren` can be invoked on a single element (represented by the variable `$one_element`) or collection of elements (represented by the expression `$one_element/*`).

```
redChildren($one_element)

redChildren($one_element/*)
```

1.10.8 Issues Relating to Use Case FNPARM

The following issues were submitted by Phil Wadler:

1.10.8.1 Interpretation of TYPE

The notion of type can be interpreted in at least two ways: (a) as any element declared to have that type (meaning the content is a forest of that type); or (b) as any forest with that type (meaning the legal content of any element declared to have that type).

1.10.8.2 Derivation by extension

Let type C be derived from type B by extension.

As discussed at XML Query meeting of 22 March 2001, allowing a value of type C to be passed where a value of type B is expected raises many questions as to the intended interpretation. For the particular example listed, there is the question of what happens if `Address` is extended with an additional `street` element.

As pointed out by Michael Sperberg-McQueen, another possible way to deal with derivation by extension is to only allow type C to be passed where a value of type B is expected if C is derived from B by restriction, not extension.

1.10.8.3 Anytype

The interpretation of `TYPE` given in Subcase 1 seems to be inconsistent with the interpretation given in Subcase 7. According to Subcase 1, `TYPE AnyElement` should only allow one to pass any element with content declared to be `AnyElement`, not any element whatsover.

Editor's note: This issue may have been resolved by the keyword MATCH in Subcase 7.

1.10.8.4 Overall

The proposal has no way to describe arguments other than single elements or lists. For instance, there is no way to say that the argument to a function is a forest consisting of two name elements, or a forest containing a name, street, and city, or whatever. This capability is useful and is present in the algebra, we should consider whether to include it in XQuery.

A Acknowledgements

The editors thank the members of the XML Query Working Group, which produced the material in this document.

The use cases in this paper were contributed by the following individuals:

Use Case "R"	Don Chamberlin
Use Case "XMP"	Mary Fernandez, Jerome Simeon, Phil Wadler
Use Case "TREE"	Jonathan Robie
Use Case "PARTS"	Michael Rys
Use Case "NS"	Ingo Macherius
Use Case "REF"	Don Chamberlin
Use Case "TEXT"	Umit Yalcinalp
Use Case "SEQ"	Jonathan Robie
Use Case "SGML"	Paula Angerstein
Use Case "FNPARM"	Don Chamberlin

Use case "XMP" has been previously published in [Fernandez]. Use cases "Tree" and "Seq" have been previously published in [Robie99].

The editors also wish to thank the members of the other W3C Working Groups who have commented on earlier drafts, and Michael Dyck for his critical reading and helpful suggestions.

B References (Non-Normative)

The following references are some of the works considered by the WG in deriving its use cases.

Maier98
Database Desiderata for an XML Query Language, David Maier, 1998. In Query Languages 98 (QL'98). Available at http://www.w3.org/TandS/QL/QL98/pp/maier.html.

Cotton98
Candidate Requirements for XML Query, Paul Cotton and Ashok Malhotra, 1998. In Query Languages 98 (QL'98). Available at http://www.w3.org/TandS/QL/QL98/pp/queryreq.html.

Fernandez
XML Query Languages: Experiences and Exemplars, Mary Fernandez, Jerome Simeon, Philip Wadler, 1999. Available at http://www.w3.org/1999/09/ql/docs/xquery.html.

Robie99
The Tree Structure of XML Queries, Jonathan Robie. Available at http://www.w3.org/XML/Group/1999/10/xquery-tree.html (W3C members only)

XML
Extensible Markup Language (XML), Version 1.0. W3C Recommendation. Available at http://www.w3.org/TR/1998/REC-xml-19980210.

XPath
XML Path Language (XPath), Version 1.0. W3C Recommendation. Available at http://www.w3.org/TR/xpath.

Namespaces
Namespaces in XML. W3C Recommendation. Available at http://www.w3.org/TR/1999/REC-xml-names-19990114/.

DOM
Document Object Model (DOM), Level 2 Specification. W3C Candidate Recommendation. Available at http://www.w3.org/TR/DOM-Level-2-Core/.

XSLT
XSL Transformations (XSLT), Version 1.0. W3C Recommendation. Available at http://www.w3.org/TR/xslt.

Infoset
XML Information Set, W3C Working Draft 20-December-1999, John Cowan, David Megginson (eds.), 1999. Available at http://www.w3.org/TR/xml-infoset/ .

XMLSchema0
XML Schema Part 0: Primer, David C. Fallside (ed.), 2000. Available at http://www.w3.org/TR/xmlschema-0/

XMLSchema1
XML Schema Part 1: Structures, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelson (eds.), 2000. Available at http://www.w3.org/TR/xmlschema-1/

XMLSchema2
XML Schema Part 2: Datatypes, Paul V. Biron, Ashok Malhotra (eds.), 2000. Available at http://www.w3.org/TR/xmlschema-2/

Use Case Sample Queries
http://www.w3.org/TR/2001/WD-xmlquery-use-cases-20010608/xmlquery-use-case-queries.txt

XQuery Sample Queries
http://www.w3.org/TR/2001/WD-xquery-20010608/xquery-wd-queries.txt