Query XML data that contains namespaces

Cynthia M. Saracco

November 09, 2006

The widespread use of namespaces in XML messages and documents impacts how application developers must write their queries. Unfortunately, the relationship between namespaces and popular query languages (such as SQL/XML and XQuery) is often poorly understood. This article walks you through several common scenarios to help you learn how to query XML data that contains namespaces.

To some, they're a solution; to others, they're a stumbling block. But whether you love them or loathe them, XML namespaces are everywhere. Indeed, many developers consider them a necessity.

Open a Web service definition language (WSDL) file or SOAP message, convert a word processing document into XML, or skim through industry-specific XML schemas; chances are you'll find references to multiple namespaces. And that's something you need to consider if you want to query XML data, because namespaces change everything. (Well, not quite everything. But namespaces have a significant impact on the semantics of your queries.)

Ignore the presence of namespaces, and the queries you write aren't likely to produce the desired results. That's the bad news. The good news is that it won't take a large investment of time to learn how to properly query XML data that contains namespaces. This article can help you get started.

Introducing XML namespaces

XML namespaces are a W3C XML standard. Indeed, XPath, XML Schema, XQuery, and other XML technologies support namespaces. While this article does not provide a detailed tutorial on namespaces, it does highlight a few key concepts. To learn more about XML namespaces, see the Related topics section.

XML namespaces allow XML documents to incorporate elements and attributes from different vocabularies without ambiguity and processing conflicts. XML namespaces potentially provide organizations a universally unique markup vocabulary (element and attribute names) for their XML data, as well as the ability to share this vocabulary with others. Some firms rely on namespaces to help them combine XML data from different sources, to version XML schemas as business needs evolve, and to promote document reuse.

Still, many IT professionals disagree on how namespaces should be used. Some advocate frequent and widespread use in documents and schemas. Others urge caution or avoidance. If
you're curious about the debate, see the Related topics section. I won't jump into that fray in this article. I'll just explain how you can query XML data that contains namespaces, because you'll probably encounter namespaces in your work.

What, then, are namespaces? They're a collection of unique XML element and attribute names identified by a Uniform Resource Identifier (URI). These URIs often look like Web site URLs (that is, they include domain names such as http://www.w3.org), although sometimes a Universal Resource Name (URN) is used. In either case, a URI doesn't actually retrieve data from the specified location. If the URI takes the form of a URL, it doesn't even need to reference a real Web page; it can be a "fake" URL that simply serves as an identifier.

XML namespaces are declared using the xmlns keyword. Listing 1 shows two valid namespace declarations for elements within an XML document. The <employee> element has a URL-based URI, while the <partner> element has a URN-based URI.

Listing 1. Sample namespace declarations

```xml
<employee xmlns="http://www.bogus.com/employee/1.0"/>
<p:partner xmlns:p="urn:xmlns:bogus:partner1.0"/>
```

Did you notice another difference between these namespace declarations, beyond the type of URI specified? Indeed, the <partner> element includes a namespace that both defines and contains a prefix (p, in this example). The use of namespace prefixes is optional, but it's more than just a stylistic choice. If an element has a namespace prefix, it belongs to the namespace defined for that prefix; however, its child elements don't belong to the same namespace unless they are prefixed as well. Elements that contain a namespace declaration but no namespace prefix (such as the <employee> element in the previous example) belong to the declared namespace. Their child elements also belong to this same namespace unless specifically overridden. Finally, unprefixed elements that lack an explicit namespace declaration are bound to the in-scope default namespace. If there is no such binding, the element does not belong to any namespace.

Consider the example in Listing 2:

Listing 2. A sample XML document with multiple namespaces

```xml
<mydoc>
  <employee xmlns="http://www.bogus.com/employee/1.0">
    <name>John Jones</name>
    <employee>
      <p:partner xmlns:p="urn:xmlns:bogus:partner1.0"/>
      <name>Acme Computer Corp.</name>
    </p:partner>
    <department>
      <name>Sales</name>
    </department>
  </employee>
</mydoc>
```
In this case, the employee name belongs to the default namespace declared in the `<employee>` element (http://www.bogus.com/employee/1.0). However, the partner name isn't within the scope of any namespace. Although it's a sub-element of `<partner>`, this `<name>` element doesn't inherit its parent's namespace because that namespace is declared with a prefix. You can rewrite the line as follows to include the partner's name information in the same namespace as its parent:

### Listing 3. An element modified to include a namespace prefix

```xml
<p:name>Acme Computer Corp.</p:name>
```

Finally, the department name doesn't belong to any namespace. This is because no namespace is declared in the `<department>` element, and it isn't bound to a default namespace.

As you can see, it's easy to create confusion by mixing and matching various forms of namespace declarations within a single document. In general, if you have the luxury of defining the XML data you'll be working with, be consistent in your use of namespaces. It simplifies your applications and queries. Subsequent sections explore how the scope of a namespace impacts queries.

### Sample environment

Before you can consider how to query XML data that contains namespaces, you need some sample data. For this purpose, I'm using DB2 V9 to store and query XML data about business partners, which I'll maintain in a single PARTNERS table. If you plan to follow along, you need to create this table in a DB2 UTF-8 database. (See "Get off to a fast start with DB2 Viper" for database creation instructions.)

The script in Listing 4 creates the table and inserts several rows into this table (each of which contains one XML document):

### Listing 4. The script to create a sample table with data

```sql
CREATE TABLE partners (id INT PRIMARY KEY NOT NULL, status VARCHAR(10), details XML);

INSERT INTO partners VALUES (111, 'Gold',
  '<company type="public">
    <name>Acme Tech</name>
    <specialty>Technology</specialty>
    <contact>
      <name>John Smith</name>
      <title>VP, Business Development</title>
      <email>js@us.acme_tech.com</email>
    </contact>
  </company>');

INSERT INTO partners VALUES (200, 'Silver',
  '<company xmlns="urn:xmlns:saracco-sample:company1.0" type="public">
    <name>Saturnia Ltd</name>
    <specialty>Technology</specialty>
    <contact>
      <name>Klaus Fischer</name>
      <title>Alliance Manager</title>
      <email>klausfischer@uk.saturnia.com</email>
    </contact>
  </company>');</n```
insert into partners values (222, 'Gold',
'&lt;co:company xmlns:co="urn:xmlns:saracco-sample:company1.0" type="private"&gt;
  &lt;name&gt;Maribel Enterprises&lt;/name&gt;
  &lt;specialty&gt;Public relations&lt;/specialty&gt;
  &lt;contact&gt;
    &lt;name&gt;Maribel Payton&lt;/name&gt;
    &lt;title&gt;CEO&lt;/title&gt;
    &lt;email&gt;mpayton@maribelent.com&lt;/email&gt;
  &lt;/contact&gt;
&lt;/co:company&gt;');

insert into partners values (333, 'Silver',
'&lt;co:company xmlns:co="urn:xmlns:saracco-sample:company1.0" type="public"&gt;
  &lt;co:name&gt;Credo International Corp.&lt;/co:name&gt;
  &lt;co:specialty&gt;Manufacturing&lt;/co:specialty&gt;
  &lt;person:contact xmlns:person="http://www.ibm.com/bogus/saracco-sample/person1.0"&gt;
    &lt;person:name&gt;Helen Rakai&lt;/person:name&gt;
    &lt;person:title&gt;Director of Marketing&lt;/person:title&gt;
    &lt;person:email&gt;helen_rakai@credointcorp.com&lt;/person:email&gt;
  &lt;/person:contact&gt;
&lt;/co:company&gt;');

insert into partners values (444, 'Silver',
'&lt;co:company xmlns:co="urn:xmlns:saracco-sample:company1.0" type="private"&gt;
  &lt;co:name&gt;Raymond Associates&lt;/co:name&gt;
  &lt;co:specialty&gt;Consulting&lt;/co:specialty&gt;
  &lt;person:contact xmlns:person="http://www.ibm.com/bogus/saracco-sample/person2.0"&gt;
    &lt;person:name&gt;Raymond Sanchez&lt;/person:name&gt;
    &lt;person:title&gt;Dr.&lt;/person:title&gt;
    &lt;person:email&gt;drsanchez@ca.rrs.com&lt;/person:email&gt;
    &lt;person:comments&gt;Prefers short-term projects&lt;/person:comments&gt;
  &lt;/person:contact&gt;
&lt;/co:company&gt;');

These XML documents track similar information about business partners, including the company names, area of specialty, and business partner representatives. However, each document uses namespaces somewhat differently, and some contain multiple namespaces. I've purposefully included diverse examples of namespace usage to illustrate the impact these namespaces will have on subsequent queries.

The queries that follow are all designed to be issued from the DB2 command-line processor. From a DB2 command window, issue the following command to set up a command-line environment in which query statements terminate with a percentage sign (\%), and XML output displays in an easy-to-read fashion:

**Listing 5. Set up the DB2 command-line environment**

db2 -td% -i -d

The \% is not the default termination character. The default termination character (a semicolon or ";") must be overridden in the command-line environment because it is reserved in XQuery for separating a prolog, such as a namespace declaration, from the body of the query.

If you prefer, you can use the Command Editor of the DB2 Control Center to issue your queries.
Use XPath expressions in SQL/XML and XQuery

Query XML data that contains namespaces requires you to specify appropriate namespace information in your path expressions. Both SQL/XML and XQuery support XPath expressions that allow for navigation through an XML hierarchy. If you're not already familiar with how to incorporate path expressions into SQL/XML and XQuery, read "Query DB2 XML data with SQL/XML" and "Query DB2 XML data with XQuery."

Now, let's step through several sample tasks and explore how to write XQueries to retrieve the desired information. I'll also show you how to write SQL/XML queries that meet your needs.

Case 1: Return all XML "company" data

Your first task is simple: retrieve "company" data about all business partners. If you want to obtain a set of all partner XML data, you could write a simple SQL query:

Listing 6. An SQL query to retrieve all company data

```
select details from partners %
```

But suppose you want to obtain a sequence of company data. To do that, you need to write an XQuery. If you've never worked with namespaces, you might be tempted to write a query such as:

Listing 7. An incorrect XQuery to retrieve all company data

```
xquery db2-fn:xmlcolumn('PARTNERS.DETAILS')/company %
```

However, this query returns only one XML document from the sample table:

Listing 8. The output from the previous XQuery

```
<company type="public">
  <name>Acme Tech</name>
  <specialty>Technology</specialty>
  <contact>
    <name>John Smith</name>
    <title>VP, Business Development</title>
    <email>js@us.acme_tech.com</email>
  </contact>
</company>
```

This is because the path expression in the query targets only <company> elements that have no namespaces. To retrieve all <company> elements, you need to rewrite the query. To do this
simply, use a wildcard (*) in the path expression. The following XQuery causes DB2 to retrieve all <company> elements at the root node of your documents, regardless of namespaces:

**Listing 9. A revised XQuery to retrieve all company data**

```xml
xquery db2-fn:xmlcolumn('PARTNERS.DETAILS')/*:company %
```

In SQL/XML, this query could be expressed as:

**Listing 10. An SQL/XML query to retrieve all company data**

```sql
select xmlquery('$c/*:company' passing details as "c")
from partners
where xmlexists('$c/*:company' passing details as "c")%
```

You may wonder why a WHERE clause appears in this query. Strictly speaking, given the sample data, it’s not necessary. That’s because every XML document in the PARTNERS table contains a root <company> element, and you want to retrieve all company information, regardless of namespaces. However, if you have a document with a root <firm> element stored in PARTNERS.DETAILS and you omit the WHERE clause shown, DB2 returns an empty record for that document. This is due to the semantics of SQL: without a WHERE clause, SQL queries do not filter out any rows from the table in the returned result set. Thus, if you use SQL/XML to query XML data, you must include a WHERE clause with an XMLExists() function (or other filtering predicate) to ensure your results don’t include a row for every row present in the table.

**Case 2: Return all XML "company" data for a selected namespace**

Often, you may need to restrict queries of XML data to specific namespaces. This section considers how to obtain full XML documents for company records associated with the urn:xmlns:saracco-sample:company1.0 namespace.

With DB2, you must declare the namespace of interest to you as part of your query. Listing 11 declares a default namespace for a subsequent XQuery expression:

**Listing 11. Declare a default namespace in an XQuery**

```xml
xquery declare default element namespace "urn:xmlns:saracco-sample:company1.0";
```

This clause cannot be run independently. Attempts to do so produce an SQL16002N error. The namespace declaration must be immediately followed by the XQuery you wish to issue. This example declares a default namespace and instructs DB2 to retrieve information about all companies associated with that namespace:

**Listing 12. An XQuery using a default namespace**

```xml
xquery declare default element namespace "urn:xmlns:saracco-sample:company1.0";
db2-fn:xmlcolumn('PARTNERS.DETAILS')/company%
```

Given the contents of Listing 4, DB2 returns a sequence of four XML records:
Listing 13. The output from the previous XQuery

```xml
<company xmlns="urn:xmlns:saracco-sample:company1.0" type="public">
  <name>Saturnia Ltd</name>
  <specialty>Technology</specialty>
  <contact>
    <name>Klaus Fischer</name>
    <title>Alliance Manager</title>
    <email>klausfischer@uk.saturnia.com</email>
  </contact>
</company>

<co:company xmlns:co="urn:xmlns:saracco-sample:company1.0" type="private">
  <name>Maribel Enterprises</name>
  <specialty>Public relations</specialty>
  <contact>
    <name>Maribel Payton</name>
    <title>CEO</title>
    <email>mpayton@maribelent.com</email>
  </contact>
</co:company>

<co:company xmlns:co="urn:xmlns:saracco-sample:company1.0" type="public">
  <co:name>Credo International Corp.</co:name>
  <co:specialty>Manufacturing</co:specialty>
  <person:contact xmlns:person="http://www.ibm.com/bogus/saracco-sample/person1.0">
    <person:name>Helen Rakai</person:name>
    <person:title>Director of Marketing</person:title>
    <person:email>helen_rakai@credointcorp.com</person:email>
  </person:contact>
</co:company>

<co:company xmlns:co="urn:xmlns:saracco-sample:company1.0" type="private">
  <co:name>Raymond Associates</co:name>
  <co:specialty>Consulting</co:specialty>
  <person:contact xmlns:person="http://www.ibm.com/bogus/saracco-sample/person2.0">
```

Query XML data that contains namespaces
Information for Acme Tech isn't included in the result because its `<company>` element doesn't belong to the declared namespace.

Listing 14 shows one way to express the previous XQuery in SQL/XML:

**Listing 14. An SQL/XML query with a default namespace**

```sql
select xmlquery('declare default element namespace
    "urn:xmlns:saracco-sample:company1.0";
    $c/company' passing details as "c")
from partners
where xmlexists('declare default element namespace
    "urn:xmlns:saracco-sample:company1.0";
    $c/company' passing details as "c") %
```

The `XMLExists()` function restricts the results to the four company records associated with the namespace of interest.

**Case 3: Explore case sensitivity in namespaces**

Consider a query very similar to the example shown in Listing 12:

**Listing 15. Revised XQuery with a namespace modification**

```xml
xquery declare default element namespace "urn:xmlns:saracco-sample:Company1.0";
```

On close inspection of this query, only a single character differs from the XQuery shown earlier. In this case, "Company1.0" begins with a capital letter, while the previous query referred to "company1.0" as part of the namespace definition. This query executes successfully but returns no records. That's because namespaces are case sensitive, as are XPath expressions.

If your queries execute without error but return no data, double-check the path expressions and namespace declarations in your query. This is true for both XQuery and SQL/XML.

**Case 4: Declare prefixed namespaces**

Until now, the examples declared a default namespace for each of the queries. However, you can also declare a namespace with a prefix and reference this prefix in your queries as needed. If
you typically create XML documents that contain namespace prefixes, this approach will be quite natural to you. Furthermore, if your query needs to reference XML elements that belong to different namespaces, you have to use prefixes, as you'll see later in this article.

Here's how to rewrite the XQuery in Listing 12 to use a namespace prefix rather than a default namespace:

**Listing 16. Use a namespace prefix in an XQuery**

```xml
xquery declare namespace x="urn:xmlns:saracco-sample:company1.0";
db2-fn:xmlcolumn('PARTNERS.DETAILS')/x:company
```

Similarly, here's how to rewrite the SQL/XML equivalent to use a namespace prefix:

**Listing 17. The SQL/XML equivalent to the previous XQuery**

```sql
select xmlquery('declare namespace x="urn:xmlns:saracco-sample:company1.0";
  $c/x:company' passing details as "c")
from partners
where xmlexists('declare namespace x="urn:xmlns:saracco-sample:company1.0";
  $c/x:company' passing details as "c")
```

The namespace prefix used in the query can be different from the prefix used in the data. What matters is that the prefix is bound to the same namespace URI that is used in the data. You can use a prefixed namespace in your query to retrieve elements that have a default namespace in your documents, as well as perform the reverse operation.

**Case 5: Retrieve XML fragments**

While cases 1 through 4 retrieved entire XML documents stored in DB2, it's quite common to write queries that retrieve only document fragments. Of course, the presence of namespaces impacts such queries as well.

Consider this XQuery, which instructs DB2 to retrieve the company names of all partners in which the `<company>` element and its child `<name>` element belong to a common namespace (urn:xmlns:saracco-sample:company1.0):

**Listing 18. An XQuery to retrieve company names that belong to a specific namespace**

```xml
xquery declare namespace c="urn:xmlns:saracco-sample:company1.0";
db2-fn:xmlcolumn('PARTNERS.DETAILS')/c:company/c:name %
```

This query returns a sequence of three XML elements when run against the sample data:
Listing 19. The output from the previous XQuery

```
<name xmlns="urn:xmlns:saracco-sample:company1.0">Saturnia Ltd</name>
<co:name xmlns:co="urn:xmlns:saracco-sample:company1.0">Credo International Corp.</co:name>
<co:name xmlns:co="urn:xmlns:saracco-sample:company1.0">Raymond Associates</co:name>
```

The record for Maribel Enterprises isn't returned. Although its `<company>` element belongs to the namespace specified in the query, its `<name>` element does not. This is because the record's `<company>` element is defined with a namespace prefix in the table. Because its child nodes (including the `<name>` element) don't contain namespace prefixes, they don't belong to any namespace.

Here's how to express the previous query in SQL/XML:

Listing 20. The SQL/XML equivalent to the previous XQuery

```
select xmlquery('declare namespace x="urn:xmlns:saracco-sample:company1.0";
    $c/x:company/x:name' passing details as "c")
from partners
where xmlexists('declare namespace x="urn:xmlns:saracco-sample:company1.0";
    $c/x:company/x:name' passing details as "c") %
```

Case 6: Reference multiple namespaces in a single query

Since many XML documents contain elements associated with different namespaces, some of your queries may need to reference multiple namespaces. In this case, simply declare multiple namespaces in the query and reference each as needed.

Consider this query, which retrieves contact information for various companies:

Listing 21. Use multiple namespaces in an XQuery

```
xquery declare namespace p="http://www.ibm.com/bogus/saracco-sample/person2.0";
declare namespace c="urn:xmlns:saracco-sample:company1.0";
db2-fn:xmlcolumn('PARTNERS.DETAILS')/c:company/p:contact %
```

Given the sample data in Listing 4, DB2 returns a single record that contains contact information for Raymond Sanchez:
Listing 22. The output from the previous XQuery

```xml
  <person:name>
    Raymond Sanchez
  </person:name>
  <person:title>
    Dr.
  </person:title>
    President
  </job:title>
  <person:email>
    drsanchez@ca.rrs.com
  </person:email>
  <person:comments>
    Prefers short-term projects
  </person:comments>
</person:contact>
```

These are some of the reasons additional records don't qualify:

- John Smith's record has no namespaces associated with his company or contact data.
- Klaus Fischer's record shares the specific company namespace declaration, but his contact data belongs to a namespace that isn't specified in the query. (His contact data is associated with the urn:xmlns:saracco-sample:company1.0 namespace.)
- Maribel Payton's record shares the specific company namespace declaration, but her contact data doesn't belong to any namespace.

Listing 23 shows one way to express this query in SQL/XML:

Listing 23. The SQL/XML equivalent to the previous XQuery

```sql
select xmlquery('declare namespace x="urn:xmlns:saracco-sample:company1.0";
                  declare namespace y="http://www.ibm.com/bogus/saracco-sample/person2.0";
                  $c/x:company/y:contact' passing details as "c")
from partners
where xmlexists('declare namespace x="urn:xmlns:saracco-sample:company1.0";
                  declare namespace y="http://www.ibm.com/bogus/saracco-sample/person2.0";
                  $c/x:company/y:contact' passing details as "c")
```

Case 7: More information on multiple namespaces

Since working with multiple namespaces can seem tricky at first, consider another example that's slightly more complex. Review this query and see if you can understand its intent:

Listing 24. Another XQuery that references multiple namespaces

```xml
xquery declare namespace p="http://www.ibm.com/bogus/saracco-sample/person1.0";
db2-fn:xmlcolumn('PARTNERS.DETAILS')/*:company/p:contact/title %
```

This query instructs DB2 to retrieve the `<title>` element of contacts for various companies. It also specifies that qualifying `<company>` elements can be associated with any namespace, that
<contact> elements must belong to the http://www.ibm.com/bogus/saracco-sample/person1.0 namespace, and that <title> elements cannot belong to any namespace.

Given the sample data, no records are returned for this query. One record -- for Helen Rakai -- comes close because it contains a root <company> element with a <contact> sub-element that belongs to the namespace specified. However, its <title> element belongs to a specific namespace; because of this, the record does not match the query's filtering criteria.

Case 8: Use namespaces and attributes in queries
Cases 1 through 7 involve queries over XML element nodes. You need to consider how namespaces apply to attributes as well. The situation isn't what you might expect.

Attributes never inherit a namespace from their elements and never assume a default namespace. If an attribute has a prefix, it belongs to the namespace indicated by the prefix. If an attribute has no prefix, it has no namespace. For details, see the "W3C Namespaces Recommendation". Of course, you need to take such information into account when you write queries.

Consider this XQuery example, which retrieves the names of public companies:

Listing 25. An XQuery example that involves a namespace and an attribute
```xml
xquery declare namespace c="urn:xmlns:saracco-sample:company1.0";
for $x in db2-fn:xmlcolumn('PARTNERS.DETAILS')/c:company
where $x/@type="public"
return $x/c:name %
```

This query specifies that qualifying <company> and <name> elements must belong to a specific namespace (urn:xmlns:saracco-sample:company1.0). The presence of a namespace prefix (c) in this query indicates that. Note that the @type attribute, which indicates whether a company is public or private, does not include a namespace prefix. Thus, this query produces two records:

Listing 26. The output from the previous XQuery
```xml
<name xmlns="urn:xmlns:saracco-sample:company1.0">
Saturnia Ltd
</name>
<co:name xmlns:co="urn:xmlns:saracco-sample:company1.0">
Credo International Corp.
</co:name>
```

If you specify a namespace for the @type attribute in the query, as shown in Listing 27, none of the sample XML documents qualify. In this case, DB2 returns no records:

Listing 27. A revised XQuery
```xml
xquery declare namespace c="urn:xmlns:saracco-sample:company1.0";
for $x in db2-fn:xmlcolumn('PARTNERS.DETAILS')/c:company
where $x/@c:type="public"
return $x/c:name %
```

If you want to use SQL/XML to retrieve the names of public companies, you could write a query similar to this one:
Listing 28. The SQL/XML equivalent of the XQuery shown in Listing 25

```sql
select xmlquery('declare namespace x="urn:xmlns:saracco-sample:company1.0";
 $c/x:company/x:name' passing details as "c")
from partners
where xmlexists('declare namespace x="urn:xmlns:saracco-sample:company1.0";
 $c/x:company[@type="public"]' passing details as "c") %
```

Case 9: Convert XML data to relational data

Finally, how do you convert XML data with namespaces into relational data? Many existing applications and commercial tools (such as query/report writers) are designed to work with data stored in the columns of traditional SQL data types, such as VARCHAR, INT, DATE, and so on. As a result, programmers use the `SQL/XML()` function to dynamically convert XML data into more traditional SQL data types.

Of course, the presence of namespaces in the original XML data affects how you must write such "transformational" queries. Consider the next example, which retrieves information about business partners with a "Silver" status whose company data is associated with a given namespace. In particular, the query returns the company's ID and name, as well as the name and email address of the company's contact person.

Listing 29. An SQL/XML query that involves `XMLTable()` and namespaces

```sql
select p.id, z.company, z.name, z.email
from partners p,
xmltable('declare namespace ns="urn:xmlns:saracco-sample:company1.0";
 $c/ns:company' passing details as "c"
columns
  "COMPANY" varchar(30) path
  'declare namespace ns="urn:xmlns:saracco-sample:company1.0";ns:name',
  "NAME" varchar(30) path '*:contact/*:name',
  "EMAIL" varchar(30) path '*:contact/*:email'
) as z
where p.status = 'Silver' %
```

A few aspects of this query are worth noting. First, the query is written in SQL/XML. You cannot express this particular query in pure XQuery (with no SQL) because the data to be returned includes company IDs, which are stored in an SQL integer column. XQuery operates on XML data types, not relational data types. Secondly, each path expression in the SQL/XML query requires a separate namespace declaration. Thus, this query declares the namespace for `urn:xmlns:saracco-sample:company1.0` twice in this example. Lastly, names and email addresses of qualifying company representatives can belong to any namespace (or to no namespace), so the path expressions for the NAME and EMAIL columns don't need to declare a namespace.

DB2 returns a three-row result set for this query:

Listing 30. The output from the previous SQL/XML query

<table>
<thead>
<tr>
<th>ID</th>
<th>COMPANY</th>
<th>NAME</th>
<th>EMAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Saturnia Ltd</td>
<td>Klaus Fischer</td>
<td><a href="mailto:klausfischer@uk.saturnia.com">klausfischer@uk.saturnia.com</a></td>
</tr>
<tr>
<td>333</td>
<td>Credo International Corp.</td>
<td>Helen Rakai</td>
<td><a href="mailto:helen_rakai@credointcorp.com">helen_rakai@credointcorp.com</a></td>
</tr>
<tr>
<td>444</td>
<td>Raymond Associates</td>
<td>Raymond Sanchez</td>
<td><a href="mailto:drsanchez@ca.rrs.com">drsanchez@ca.rrs.com</a></td>
</tr>
</tbody>
</table>

Query XML data that contains namespaces
Summary

If you want to learn more than the basics of XQuery and SQL/XML, you must understand how the presence of XML namespaces in documents and messages impact the semantics of their queries. If you don't, you may have unexpected, or undesired, results. Fortunately, it's not hard to learn how to query XML documents that contain namespaces. This article provides several examples to help you get started.

Acknowledgments

Thanks to Matthias Nicola, Bryan Patterson, and Bert van der Linden for their review of this paper.
Related topics

- **DB2 XML technical enablement Wiki**: Find papers, Webcasts, and demos on DB2 XML.
- "**Query DB2 XML data with XQuery**" (developerWorks, April 2006): Learn more about XQuery.
- "**Query DB2 XML data with SQL**" (developerWorks, March 2006): Learn more about SQL/XML.
- "**Get off to a fast start with DB2 Viper**" (developerWorks, March 2006): Learn more about DB2 9 pureXML.
- For a brief introduction to XML namespaces, read "**Plan to use XML namespaces, part 1**" (developerWorks, November 2002), "**XML Namespace by Example**" (O'Reilly xml.com, January 1999), or review an [online tutorial](#).
- "**W3C Namespaces Recommendation**": Get detailed information on XML namespaces.
- Build your next development project with [IBM trial software](#), available for download directly from developerWorks.