This tutorial introduces pureXML™ and XQuery, starting from a basic overview of the characteristics and advantages of the XML data type, and then compares it with standard relational tables. Users are then asked to write XQueries to retrieve XML elements, filter data based on XML values, transform XML output, and use various clauses to select data more precisely. The tutorial ends with a section on mixing XQuery and SQL. This tutorial is Part 7 of the SQL & XQuery tutorial for IBM® DB2® series. (Note: You must have DB2 9 installed to do the hands-on exercises in this tutorial.)
this tutorial to enhance their database query skills. Academic Initiative members can use this tutorial series as a part of their database curriculum.

All the examples in this document are based on Aroma, a sample database that contains sales data for coffee and tea products sold in stores across the United States. Each example consists of three parts:

- A business question, expressed in everyday language
- One or more example queries, expressed in SQL or XQuery
- A table of results returned from the database

This guide is designed to allow participants to learn the SQL language and XQuery. As with any learning, it is important to supplement it with hands-on exercises. This is facilitated by the table definitions and data.

For students using this as part of an academic class, obtain from your instructor the instructions to connect to the Aroma database and learn about any differences between the guide and your local set up.

This tutorial was written for DB2 Express-C 9 for UNIX®, Linux® and Windows® (formerly known as Viper).

About this tutorial

This tutorial introduces readers to pureXML and XQuery. The only place XML is used in the Aroma database is the Comments column. This column was mentioned in Part 1 of this series and it is the focus of this tutorial, part 7.

This tutorial starts out with a basic overview of the characteristics and advantages of the XML data type, and then compares it with standards relational tables. It then asks readers to write XQueries to retrieve XML elements, filter data based on XML values, transform XML output, and use various clauses to select data more precisely. The tutorial ends with a section on mixing XQuery and SQL and combining the power of the two languages.

Connecting to a database

You need to connect to a database before you can use SQL statements to query or manipulate data. The CONNECT statement associates a database connection with a user name.

Find out from your instructor the database name that you will need to be connected to. For this series, the database name is aromadb.

To connect to the aromadb database, type the following command in the DB2 command line processor:
CONNECT TO aromadb USER userid USING password

Replace the user ID and password with the user ID and password that you received from your instructor. If no user ID and password are required, simply use the following command:

CONNECT TO aromadb

The following message tells you that you have made a successful connection:

Database Connection Information
Database server = DB2/NT 9.0.0
SQL authorization ID = USERID
Local database alias = AROMADB

Once you are connected, you can start using the database.

Section 2. About XML

What is XML?

XML...

- is the standard for exchanging data between different systems, platforms, applications, and organizations
- is independent from vendor and platform
- is highly flexible
- is suited for any combination of structured, unstructured and semi-structured data
- is easy to extend--new tags can be defined as needed
- can easily be transformed into "different-looking" XML and even into other formats such as HTML
- can easily be checked for compliance with a schema

All this has become possible through widely available tools and standards, such as XML parsers, XSLT, and XML schema. They greatly relieve applications from the burden of dealing with peculiarities of proprietary data formats. In an era where message formats, business forms, and services change frequently, XML reduces the cost and time it takes to maintain application logic correspondingly.
Beyond XML for data exchange, enterprises are keeping large amounts of business-critical data permanently in XML format. This has various reasons:

- Some businesses must retain XML documents in their original format for auditing and regulatory compliance e.g., legal and financial documents as well as eForms, particularly in the government sector
- XML can be a more suitable data model than a relational schema. This is not only true for content-oriented applications, but also for certain data-oriented applications e.g., in life-science applications, the data is highly complex and hierarchical in nature and yet may contain significant amounts of unstructured information. Most of today's genomic data is still kept in proprietary flat file formats, but major efforts are underway to move to XML.

Relational databases have been offering support for storage, manipulation, search, and retrieval of XML data. This is usually based on storing XML documents in LOBs or mapping and shredding XML to a relational schema.

These solutions have inherent functional and performance constraints. Generally, LOB-based storage allows for fast insert and retrieval of full documents but suffers from poor search and extract performance due to XML parsing at query execution time. This can be moderately improved if indexes are built at insert time. While this incurs XML parsing overhead, it may speed up queries that look for documents that match given search conditions. Yet, extraction of document fragments and sub-document level updates still require expensive XML parsing.

In the previous parts of this tutorial series, you have read about SQL including the SELECT statement and the Data Definition Language (DDL). XML is a different way of organizing data and uses XQuery or Xpath to query the data.

Relational data is represented as rows and columns of data in a table format. XML data includes tags and data together.

Sample data

```
<Comments>
  <comment>
    <comment_ID>5301</comment_ID>
    <customer_info>
      <fname>Scott</fname>
      <lname>Phillips</lname>
      <email>Scott_Phillips@hotmail.com</email>
    </customer_info>
    <feedback>
      <type>opinion</type>
      <content>Gold Tips was highly enjoyable!</content>
    </feedback>
    <store_rating>
      <score>5</score>
      <out_of>5</out_of>
    </store_rating>
  </comment>
</Comments>
```
Based on the sample data alone, can you determine Scott Phillips' email address? Simply follow the hierarchy created by the nested tags. Since XML stores tags and data together, the data is self-descriptive and easy to understand.

DB2 9

DB2 9 is the industry's first hybrid data server for managing data from both relational and pureXML formats. DB2 has been providing high-performance data storage and access for relational data based on SQL standards and data storage optimizations such as data partitioning and advanced indexing and query optimization techniques. Now, DB2 has introduced an optimized data storage engine for XML data alongside the existing relational engine.

Application developers can now store XML data directly inside of a DB2 server and reap the benefits of transactions, advanced data resiliency, secure access, and, of course, the ability to search large amounts of XML data using XQuery.

XML data versus relational data

Three fundamental properties of XML make it different from the relational model:

- **XML is self-describing.** Documents contain not only the data, but also the necessary metadata. As a result, an XML document can be searched or updated without requiring a static definition of the schema. Relational models, on the other hand, require more static schema definitions. All the rows of a table must have the same schema.

- **XML is hierarchical.** Documents represent not only base information, but also information about the relationship of data items to each other in the
form of the hierarchy. Relational models require all relationship information to be expressed either by primary key or foreign key relationships or by representing that information in other relations.

- **XML is sequence-oriented** (order is important). Relational models are set-oriented; order is unimportant.

None of these differences indicate that XML is better or worse than purely relational models. In fact, XML and relational models are complementary solutions. Some data is inherently hierarchical, while other data is inherently tabular; some data has more rigid schema, while other data has less rigid schema; some data needs to follow a specific order while other data does not.

**When to use XML**

Situations in which an XML representation is beneficial include:

- **When schema is volatile.**
  If the schema of the data changes often, then representing it in relational form may be onerous given the cost and difficulty of changing the associated relational schema. The self-describing nature of XML makes schema modification much simpler.

- **When data is inherently hierarchical.**
  Some data is inherently tabular, and a relational model makes the most sense for it. Other data is naturally hierarchical, and XML is often the best representation for it.

- **When data represents business objects in which the component parts do not make sense when removed from the context of that business object.**
  For example, consider a standard employee and phone number relationship in which one employee can have multiple phone numbers: one for the office, one for fax, one for home, and one for mobile. If the most frequent usage pattern is to retrieve phone numbers in the context of the employee, it does not make sense to normalize the data and introduce a table solely to track the multiplicity of phone numbers associated with an employee. A better choice may be to keep those phone numbers in the employee relation and represent them using XML.

- **When applications have sparse attributes.**
  Some applications have a large number of possible attributes, most of which are sparse or absent for any given data value. A classic example is a merchant catalog; the number of different attributes to track for a given catalog item is huge, including size, color, weave, power requirements and a nearly endless list of other considerations. For any given object, only a subset of these attributes is relevant; the weave of a sweater makes sense but the weave of a lawn mower is nonsensical. Using a relational table to describe the characteristics of the object can be costly and overly complex. Representing such descriptive attribute information
as XML data allows for more natural representation as well as less complex and expensive searching.

- When low-volume data is highly structured.
  In many applications, structured information is critical to the application but it exists in very small quantities. While that information can be represented in a normal relation, this approach can lead to massive relational schemas. Using an XML column with multiple views can dramatically reduce the number of managed objects in a database, and thus reduce the cost of ownership.

Section 3. Creating a database that supports XML

In Part 1 of this series, you created your Aroma database with a batch file. Since you might wish to create your own databases in the future, this section teaches you how to create one that supports XML features.

Creating a database with the Control Center

Within the DB2 Control Center application, right-click on the All Databases folder on the left menu. Select Create Database -> Standard. The Create Database Wizard will pop up. Fill in the name of your database and select Enable database for XML.

XML codeset
Click **Next** twice to get to the Region screen. Set the Country/Region value to the appropriate value and change the Code set value to **UTF-8**. Click **Finish** to complete creating your database.

**Creating a database with the Command Line Processor**

To create the same database using DB2 CLP, enter the following code:

```
db2 create db aromadb using codeset utf-8 territory us
```

Note the specification of UTF-8 as the codeset for the database. The use of XML features is restricted to a database that is defined with codeset UTF-8 and has only one database partition. When creating your own databases in the future, don’t forget to include this parameter.

---

**Section 4. About the comments column**

In the Aroma database, the Sales table has a column called Comments.
It contains the following information:

- Comment ID
- Customer information: name, contact info (phone and/or email)
- Store response (if required)
- Customer feedback: could be either opinion, suggestion or question
- A store rating out of 5

Sample data

```xml
<Comments>
  <comment>
    <comment_ID>5301</comment_ID>
    <customer_info>
      <fname>Scott</fname>
      <lname>Phillips</lname>
      <email>Scott_Phillips@hotmail.com</email>
    </customer_info>
    <feedback>
      <type>opinion</type>
      <content>Gold Tips was highly enjoyable!</content>
    </feedback>
    <store_rating>
      <score>5</score>
      <out_of>5</out_of>
    </store_rating>
    <store_response>
      <required>no</required>
    </store_response>
  </comment>
  <comment>
    <comment_ID>5302</comment_ID>
    <customer_info>
      <fname>Barbara</fname>
      <lname>Adams</lname>
      <phone>6138617611</phone>
    </customer_info>
    <feedback>
      <type>question</type>
      <content>What are the top teas brands?</content>
    </feedback>
    <store_rating>
      <score>4</score>
      <out_of>5</out_of>
    </store_rating>
    <store_response>
      <required>yes</required>
      <completed>yes</completed>
      <action>Please see our "All About Tea" page at www.aroma.com/tea</action>
    </store_response>
  </comment>
</Comments>
```

Why XML?

The advantages of XML have already been discussed in earlier sections, but why choose XML over relational table columns in this specific case?
• Flexibility:
  - stores may wish to ask for different information from the customers
  - over time, stores may improve their methods of obtaining feedback, thus
    resulting in different types of data
  - e.g., promotion surveys could be different from regular sales surveys;
    this change could easily be stored with XML but not relational

• Agility:
  - table structure is not required to change along with data

• Self-descriptive:
  - store managers/employees can understand the customer comments in
    the XML form without additional explanation

• Hierarchy:
  - data is hierarchical in nature
  - customer information is meaningless if taken out of the context of the
    comment

Example questions

• What comments are stored in the Aroma database?
• What comments were received in New York stores?
• What comments require store response? Which of these have not yet
  been completed?
• Display all uncompleted comments in an HTML list.
• Display all comments and group them in the three different categories
  (comment, suggestion, and question).
• Count and display the number of comments of each type.
• Calculate and display the average rating given by all comments.

Usage notes

In the Sales database, the comments column only has value for those rows starting from March, 2006.

Section 5. About XQuery

In the previous examples, the SQL SELECT statement was used to extract data. Queries involving XML data can be done with its own query language called XQuery.
XQuery differs from SQL in a number of key respects, largely because the languages were designed to work with different data models that have different characteristics. XML documents contain hierarchies and possess an inherent order. Tabular data structures supported by SQL-based DBMSs are flat and set based; as such, rows are unordered.

The differences between these data models result in a number of fundamental differences in their respective query languages. The table below lists a few examples.

<table>
<thead>
<tr>
<th>XQuery</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>supports path expressions to enable programmers to navigate through XML's hierarchical structure</td>
<td>does not support path expressions</td>
</tr>
<tr>
<td>supports both typed and untyped data</td>
<td>is always defined with a specific type</td>
</tr>
<tr>
<td>lacks null values because XML documents omit missing or unknown data</td>
<td>uses nulls to represent missing or unknown data values</td>
</tr>
<tr>
<td>returns sequences of XML data</td>
<td>returns result sets of various SQL data types</td>
</tr>
</tbody>
</table>

Two kinds of XQuery expressions

This tutorial focuses on two important kinds of XQuery expressions: "FLWOR" expressions and path expressions.

A FLWOR expression is much like a SELECT-FROM-WHERE expression in SQL -- it is used to iterate through a list of items and to optionally return something that is computed from each item. A path expression, on the other hand, navigates through a hierarchy of XML elements and returns the elements that are found at the end of the path.

Like a SELECT-FROM-WHERE expression in SQL, an XQuery FLWOR expression may contain several clauses that begin with certain keywords. The following keywords are used to begin clauses in a FLWOR expression:

1. `for`
2. `let`
3. `where`
4. `order by`
5. `return`

A path expression in XQuery consists of a series of "steps," separated by slash characters. In its simplest form, each step navigates downward in an XML hierarchy to find the children of the elements returned by the previous step. Each step in a path expression may also contain a predicate that filters the elements that are
returned by that step, retaining only elements that satisfy some condition.

For example, for the sample data given in the previous section, assume that the variable $comments is bound to a list of XML documents containing <comment> elements, then the four-step path expression

$comments/comment/feedback[type = "suggestion"]/content

will return the list of contents for comments that are of type "suggestion".

In many cases, it is possible to write a query by using either a FLWOR expression or a path expression.

---

Section 6. Retrieving XML elements

Question

What customer comments are stored in the Aroma database?

Example FLWOR query

```xquery
for $y in db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')
return $y
```

Example path query

```xquery
db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')
```

Two queries, same result

```xml
<Comments>
  <comment>
    <comment_ID>1</comment_ID>
    <customer_info>
      <fname>Christopher</fname>
      <lname>Davis</lname>
      <phone>4147301265</phone>
      <email>Christopher.D@hotmail.com</email>
    </customer_info>
    <feedback>
      <type>opinion</type>
      <content>The employee named Heather Gray was very helpful.</content>
    </feedback>
    <store_rating>
      <score>5</score>
      <out_of>5</out_of>
    </store_rating>
  </comment>
</Comments>
```
<store_response><required>no</required></store_response>
</comment>
</Comments>

<Comments>
  <comment>
    <comment_ID>101</comment_ID>
    <customer_info>
      <fname>Cynthia</fname>
      <lname>Thomas</lname>
      <email>CThomas@hotmail.com</email>
    </customer_info>
    <feedback>
      <type>question</type>
      <content>What countries are major tea consumers?</content>
    </feedback>
    <store_rating>
      <score>4</score>
      <out_of>5</out_of>
    </store_rating>
    <store_response>
      <required>yes</required>
      <completed>yes</completed>
      <action>Please see our "All About Tea" page at www.aroma.com/tea</action>
    </store_response>
  </comment>
</Comments>

Note: The result has been formatted to make it easier for you to read. The DB2 Command Editor displays each comment on one line.

Tip: To free up memory for the Command Editor, right-click in the results area and select Clear Results.

About the queries

To execute an XQuery directly in DB2 9, you must preface the query with the keyword xquery. This instructs DB2 to invoke its XQuery parser to process your request.

Note: You only need to do this if you are using XQuery as the outermost (or top-level) language. If you embed XQuery expressions in SQL, you don't need to preface them with the xquery keyword.

When running as a top-level language, XQuery needs to have a source of input data. One way in which an XQuery can obtain input data is to call a function named db2-fn:xmlcolumn with a parameter that identifies the table name and column name of an XML column in a DB2 table.

The db2-fn:xmlcolumn function returns the sequence of XML documents that is stored in the given column. The example queries return a sequence of XML documents containing customer comment information:
Note that the column name *Comments* and table name *aroma.sales* are both specified in uppercase here. This is because they are typically folded into uppercase before being written to DB2's internal catalog. Because XQuery is case-sensitive, lowercase table and column names would fail to match uppercase names in the DB2 catalog.

---

### Section 7. Retrieving a specific XML elements

**Question**

What customer comments are stored in the Aroma database? Display only the content of the comments.

**Example FLWOR query**

```xquery
for $y in db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment/feedback/content
return $y
```

**Example path query**

```xquery
db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment/feedback/content
```

**Two queries, same result**

```xml
<content>The employee named Heather Gray was very helpful.</content>
<content>What countries are major teas consumers?</content>
<content>Demitasse Ms was highly enjoyable!</content>
<content>Xalapa Lapa was great!</content>
<content>Aroma should consider selling Jamaican Butter Rum.</content>
<content>Darjeeling Special was not enjoyable at all.</content>
<content>Special Tips is my favorite.</content>
...```

**About the queries**

The two queries, although producing the same result, are executed slightly differently.

For the FLWOR query, the second line instructs DB2 to iterate through the content sub-elements of the `<Comments>` elements contained in the *aroma.sales.comments* column. Each content element is bound in turn to the variable `$y`. The third line
indicates that, for each iteration, the value of $y$ is returned. The result is a sequence of XML elements.

For the path query, the first step calls the `db2-fn:xmlcolumn` function to obtain a list of XML documents from the `Comments` column of the `aroma.sales` table. The second step returns all the `<Comments>` elements in these documents, the third step returns the `<comment>` elements nested inside these `<Comments>` elements, the fourth step returns the `<feedback>` elements nested inside the `<comment>` elements, and the fifth step returns the `<content>` elements nested inside the `<feedback>` elements.

---

**Section 8. The text() function**

**Question**

What customer comments are stored in the Aroma database? Display the content of the comments as text.

**Example FLWOR query**

```
for $y in db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment/feedback/content
return $y/text()
```

**Example path query**

```
db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment/feedback/content/text()
```

**Two queries, same result**

The employee named Heather Gray was very helpful.
What countries are major teas consumers?
Demitasse Ms was highly enjoyable!
Xalapa Lapa was great!
Aroma should consider selling Jamaican Butter Rum.
Darjeeling Special was not enjoyable at all.
Special Tips is my favorite.
...

**About the queries**

The queries invoke the `text()` function to return a text representation of the XML
element values.

Section 9. Filtering on XML element values

Question

What customer suggestions are stored in the Aroma database? Display them as text.

Example FLWOR query

```xml
for $y in db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment/feedback
where $y/type = "suggestion"
return $y/content/text()
```

Example path query

```xml
db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment/feedback[type = "suggestion"]/content/text()
```

Two queries, same result

Aroma should consider selling Jamaican Butter Rum.
Aroma should consider selling Gyokuro Asahi Pearl Dew.
Aroma should consider selling Orange Pekoe.
Aroma should consider selling Vanilla Creme.
Aroma should consider selling China Yunnan.
Aroma should consider selling Kona Fancy 100%.
Aroma should consider selling Blackcurrant.
Aroma should consider selling Sencha.
...

About the queries

The XQuery `where` clause is similar to the SQL WHERE clause. In the FLWOR query, it filters results based on the value of the type element in the XML documents.

The `for` clause binds the variable $y to each feedback in turn. The `where` clause contains a small path expression that navigates from each feedback to its nested type element. The `where` clause is true (and the content is returned) only if the value of this type element is equal to "suggestion".
An additional predicate, [type = "suggestion"], is used to create the corresponding path query.

Question

What customer suggestions and questions are stored in the Aroma database? Display them as text.

Example FLWOR query

```
xquery
for $y in db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment/feedback
where $y/type = "suggestion" or $y/type = "question"
return $y/content/text()
```

Example path query

```
xquery
db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment/feedback[type = "suggestion" or type = "question"]/content/text()
```

Two queries, same result

What countries are major tea consumers?
Aroma should consider selling Jamaican Butter Rum.
Aroma should consider selling Gyokuro Asahi Pearl Dew.
Who do you buy your tea from?
What is the maximum amount of coffee one person can safely consume per day?
Aroma should consider selling Orange Pekoe.
Aroma should consider selling Vanilla Creme.
...

About the queries

Just like with SQL, search conditions can be used to refine the selection of XML elements. Refer back to Part 3, Using AND, NOT, and OR connectives to create complex conditions, for a detailed discussion of search conditions and logical operators.

Section 10. Transforming XML output
Question

What store responses are yet to be completed? Display the action text as an HTML list.

Example query

```xquery
<ul>
  {for $y in db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment/store_response
    where $y/completed = "no"
    return <li>{$y/action/text()}</li>}
</ul>
```

Result

```
<ul>
  <li>Thank you for your excellent suggestion! Aroma Coffee and Tea Company is actually planning on ordering the product you've suggested--check our stores next month!</li>
  <li>Thank you for your excellent suggestion! Aroma Coffee and Tea Company supports local growers in North America as well as imports. Our mission is to offer the best-quality coffee and tea products from North America as well as around the world. All of our suppliers must meet all of our quality control standards.</li>
  <li>Thank you for your excellent suggestion! Currently, Aroma Coffee and Tea Company does not plan to include the product you've suggested in our purchase plans. We will keep your suggestion on file and we will consider it in the future.</li>
  <li>Thank you for your excellent suggestion--we completely agree! Please visit www.aroma.com.</li>
  <li>Thank you for your excellent suggestion--we completely agree! Please visit www.aroma.com.</li>
  <li>Please see our "All About Coffee" page at www.aroma.com/coffee</li>
  <li>Please see our "All About Tea" page at www.aroma.com/tea</li>
  ...
</ul>
```

Note: The result has been formatted to make it easier for you to read. The DB2 Command Editor displays the entire list on one line.

About the query

A powerful aspect of XQuery is its ability to transform XML output from one form of XML into another. For example, you can use XQuery to retrieve all or part of your stored XML documents and convert the output into HTML for easy display in a Web browser. The example query retrieves and converts the store responses into XML elements that are part of an unordered HTML list.

The second line of the query causes the HTML markup for an unordered list (\texttt{<ul>}) to be included in the results. It also introduces a curly bracket, the first of two sets used in this query. Curly brackets instruct DB2 to evaluate and process the enclosed expression rather than treat it as a literal string.
The third line iterates over store responses, binding the variable \$y\) to each \store_response\ element in turn. The \return\ clause indicates that the \store_response\ elements are to be surrounded by HTML list item tags before return. The final line concludes the query and completes the HTML unordered list tag.

Section 11. The if-then-else expression

Question

What store responses are yet to be completed? Display the comment ID, customer information and store action as XML elements, under \<suggestion\> and \<question\> elements, accordingly.

Example query

```
xquery
for $y in db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment
where $y/store_response/completed = 'no'
return
  (if ($y/feedback/type = 'suggestion')
    then
      <suggestion>
        {$y/comment_ID,
         $y/customer_info,
         $y/store_response/action}
      </suggestion>
    else
      <question>
        {$y/comment_ID,
         $y/customer_info,
         $y/store_response/action}
      </question>
  )
```

Result

```
<suggestion>
  <comment_ID>4201</comment_ID>
  <customer_info>
    <fname>Karen</fname>
    <lname>Richardson</lname>
    <phone>3546388558</phone>
    <email>K.Richardson@hotmail.com</email>
  </customer_info>
  <action>Thank you for your excellent suggestion! Aroma Coffee and Tea Company is actually planning on ordering the product you've suggested--check our stores next month!</action>
</suggestion>
```
About the query

The ability of XQuery to transform XML output can be combined with its built-in support for conditional logic to reduce the complexity of application code. The query creates complex XML elements containing information about the comment ID, customer information, and store action, under the root elements <suggestion> and <question>.

The result could be useful to the person performing store actions, who would have just enough information needed to do the job.

Important: The <comment_ID> element must be capitalized exactly as shown, because XQuery is case sensitive and would not recognize it otherwise.

Section 12. The Order By clause

Question

What customer opinions are stored in the Aroma database? Order them by store rating, from lowest to highest.
Example query

```xquery
for $y in db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment
where $y/feedback/type = "opinion"
order by $y/store_rating/score
return
  <comment>
    <rating>{$y/store_rating/score/text()}</rating>
    <opinion>{$y/feedback/content/text()}</opinion>
  </comment>
```

Result

```xml
<comment>
  <rating>2</rating>
  <opinion>Darjeeling Special was not enjoyable at all.</opinion>
</comment>

... 

<comment>
  <rating>3</rating>
  <opinion>Good store, but quality of customer service could be better.</opinion>
</comment>

... 

<comment>
  <rating>4</rating>
  <opinion>The employee named Heather Gray was very helpful.</opinion>
</comment>

... 
```

Note: The result has been formatted to make it easier for you to read. The DB2 Command Editor displays each result on one line.

About the query

The `order by` clause specifies that results must be returned in ascending order (the default order) based on store ratings.

---

Section 13. The Let clause

Question

How many customer opinions are stored in the Comments column? How many
suggestions? How many questions?

Example query

```xquery
for $t in distinct-values
db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment/feedback/type
let $tc := db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment[feedback/type = $t]
return
  <comments>
    <type>{$t}</type>
    <count>{count($tc)}</count>
  </comments>
```

Result

```xml
<comments>
  <type>opinion</type>
  <count>1220</count>
</comments>
<comments>
  <type>question</type>
  <count>364</count>
</comments>
<comments>
  <type>suggestion</type>
  <count>251</count>
</comments>
```

Note: The results have been formatted to make it easier for you to read. The DB2 Command Editor displays each result on one line.

About the query

The query counts how many comments are stored under each type.

The `let` clause is used to assign a value (possibly containing a list of several items) to a variable that can be used in other clauses of the FLWOR expression.

The `distinct-values` function in the `for` clause returns a list of all the distinct values of type that are found inside `<comments>` in the `Comments` column. There are three distinct types: opinion, suggestion and question. The `for` clause binds variable `$t` to each of these type values in turn. For each value of `$t`, the `let` clause scans the `Comments` column again and binds the variable `$tc` to a list containing all the comments whose type matches the type in `$t`. The `return` clause constructs a new `<comments>` element for each distinct type value. Each of these `<comments>` elements contains two sub-elements: a `<type>` element containing the type value and a `<count>` element containing a count of how many comments are of the specified type.

The count function is one of the many built-in functions provided by XQuery. More
examples of built-in functions are given in the next section.

Section 14. Built-in functions

Question

What is the average store rating given in all comments?

Example query

```
xquery
avg(db2-fn:xmlcolumn('AROMA.SALES.COMMENTS')/Comments/comment/store_rating/score)
```

Result

```
4.11389645776567
```

About the query

The `db2-fn:xmlcolumn` function returns all scores stored in the `Comments` column as a sequence, and the `avg` function calculates their average.

This table illustrates some common XQuery functions used in business queries:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sum(sequence-expression)</code></td>
<td>Calculates the sum of all the values in sequence.</td>
</tr>
<tr>
<td><code>avg(sequence-expression)</code></td>
<td>Calculates the average of all the values in sequence.</td>
</tr>
<tr>
<td><code>max(sequence-expression)</code></td>
<td>Determines the maximum value in sequence.</td>
</tr>
<tr>
<td><code>min(sequence-expression)</code></td>
<td>Determines the minimum value in sequence.</td>
</tr>
<tr>
<td><code>count(sequence-expression)</code></td>
<td>Counts the number of non-null values in sequence.</td>
</tr>
</tbody>
</table>
Section 15. XQueries with embedded SQL

Question

What comments were left in stores in New York? Display the contents as text.

Example query

```xquery
for $y in
db2-fn:sqlquery('SELECT Comments
    FROM aroma.sales a, aroma.store b
    WHERE city = ''New York''
    AND a.storekey = b.storekey')/Comments/comment/feedback/content/text()
return $y
```

Result

Does drinking too much tea cause any health problems?
How to differentiate from imitations?
Is there a maximum amount of tea one person can safely consume per day?
Aroma should consider selling Indonesian House Roast.
Darjeeling Special was very good.
Irish Breakfast was great!
...

About the query

This query embeds SQL within XQuery to restrict results based on SQL data values. In place of the `db2-fn:xmlcolumn` function, which returns all the XML data in a column of a table, it calls the `db2-fn:sqlquery` function, which executes an SQL query and returns only the selected data. The SQL query passed to `db2-fn:sqlquery` must return XML data. This XML data can then be further processed by XQuery.

The SQL part of the query joins the `Sales` table with the `Store` table. This retrieves the rows in the `Sales` table with the same `storekey` as a New York store. The `Comments` documents stored in these rows serve as inputs to a path expression that returns all the nested content elements as text.

Important: The WHERE clause of the SQL query compares the value of `city` to the string "New York." In SQL, such strings are surrounded by single quotes. Note that although the example may appear to use double quotes, it actually
uses two single quotes before and after the comparison value ("New York"). The "extra" single quotes are escape characters. If you use double quotes around your string-based query predicate, instead of pairs of single quotes, you'll get a syntax error.

Section 16. Summary

Summary

This tutorial gave users a basic overview of XML and XQuery. It addressed the following questions: What is XML? What advantages does it have? When should it be used?

It then showed readers how to issue basic XQueries to retrieve XML elements, filter data, transform data, format data, and work with data.

XQuery and SQL are not mutually exclusive; they can be combined to issue powerful queries.

This tutorial is only meant to serve as a basic introduction to XML and XQuery. To learn more about XQueries, look it up in the DB2 Information Center or see the other learning links provided in the Resources section.

This tutorial is the last part of this series. Thank you once again for choosing IBM DB2 9 to learn about hybrid data servers serving data from both relational and pureXML structures.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
<th>Download method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aroma Database</td>
<td>Aroma_Data.zip</td>
<td>1MB</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

Information about download methods
Resources

Learn

- View this article series' "Appendix A" (developerWorks, August 2006).
- Read "DB2 XML evaluation guide" (developerWorks, June 2006), a step-by-step tutorial introducing the reader to the DB2 Viper data server on Windows platforms using the XML storage and searching (SQL/XML, XQuery) capabilities available to support next-generation applications.
- Check out this article and "Get off to a fast start with DB2 Viper" (developerWorks, March 2006).
- Learn how to "Query DB2 XML data with XQuery" (developerWorks, April 2006).
- Learn how to "Query DB2 XML data with SQL" (developerWorks, March 2006).
- Read the IBM Systems Journal and celebrate 10 years of XML.
- Refer to the SQL Reference, Vol 1 for additional information.
- Refer to the SQL Reference, Vol 2 for additional information.
- Refer to the DB2 information Center for troubleshooting.
- Visit the DB2 XML technical enablement space for links to more than 25 papers on DB2 XML capabilities.

Get products and technologies

- Now you can use DB2 for free. Download DB2 Express-C, a no-charge version of DB2 Express Edition for the community that offers the same core data features as DB2 Express Edition and provides a solid base to build and deploy applications.

Discuss

- Participate in the discussion forum for this content.
- Visit the DB2 9 On-line Support Forum.

About the authors

Pat Moffatt
Pat Moffatt is the Information Management Program Manager for the IBM Academic Initiative. Through the Academic Initiative program, she ensures that appropriate Information Management resources are made available to help faculty integrate Information Management software into their curriculum. To learn more about this program, visit www.ibm.com/university/data.
Bruce Creighton
Bruce Creighton is a Skills Segment Planner in the Information Management Education Planning and Development department. In this role, he plans investment in educational content and balances the investment between areas where IBM can attain revenue and those where the requirement for skills development are important enough to provide free education.

Jessica Cao
Jessica Cao is an Arts and Science and Computer Science student at McMaster University. She expects to complete her combined honours degree in April 2009. Jessica is working in IBM Toronto lab's DB2 Information Management Skills Channel Planning and Enablement Program to take advantage of her interest in programming, editing, and writing.

Trademarks
IBM, DB2, Universal Database, OS/2, and pureXML are registered trademarks of IBM Corporation in the United States and other countries. Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both. UNIX is a registered trademark of The Open Group in the United States and other countries. Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.