Compliments of IBM

DB2® pureXML™ FOR DUMMIES®

IBM Limited Edition

pureXML — the best native XML storage solution in the market today!

A Reference for the Rest of Us!®

FREE eTips at dummies.com®

Conor O'Mahony

DB2® pureXML™ FOR DUMMIES®

IBM LIMITED EDITION

by Conor O'Mahony

Wiley Publishing, Inc.

DB2® pureXML™ For Dummies®, IBM Limited Edition
Published by
Wiley Publishing, Inc.
111 River Street
Hoboken, NJ 07030-5774

Published by Wiley Publishing, Inc., Indianapolis, Indiana

ISBN: 978-0-470-44057-5

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

We're proud of this book; please send us your comments through our Dummies online registration form located at `www.dummies.com/register/`.

Some of the people who helped bring this book to market include the following:

Acquisitions, Editorial, and Media Development

Project Editor: Carrie A. Burchfield

Acquisitions Editor: Katie Feltman

Business Development Representative: Sue Blessing

Editorial Manager: Rev Mengle

Custom Publishing Project Specialist: Michael Sullivan

Composition Services

Project Coordinator: Kristie Rees

Layout and Graphics: Carl Byers, Reuben W. Davis, Andrea Hornberger

Proofreader: John Greenough

Publishing and Editorial for Technology Dummies

Richard Swadley, Vice President and Executive Group Publisher

Andy Cummings, Vice President and Publisher

Mary Bednarek, Executive Director, Acquisitions

Mary C. Corder, Editorial Director

Publishing for Consumer Dummies

Diane Graves Steele, Vice President and Publisher

Composition Services

Gerry Fahey, Vice President of Production Services

Debbie Stailey, Director of Composition Services

About the Author

Conor O'Mahony leads XML product strategy and messaging for IBM Information Management. You can find him blogging at `www.nativeXMLdatabase.com`. Conor is a frequent speaker at conferences and Web casts. Previously, Conor developed product strategy and brought IBM products to market for compliance, legal discovery, text analytics, text mining, classification, and search. Conor O'Mahony is based in Belmont, Massachusetts. You can reach him at `conor@us.ibm.com`.

Author's Acknowledgements

Heartfelt thanks to Brenda Brown, Bryan Patterson, Camille Cannistraci, Christy Maver, Cindy Saracco, Henrik Loeser, Irshad Raihan, Mary Desisto, and Matthias Nicola for their contributions and assistance.

Contents at a Glance

Table of Contents

Introduction

Welcome to *DB2 pureXML For Dummies.* We're very excited about this topic and hope that our enthusiasm is contagious. We believe that pureXML is the best native XML storage solution in the market today.

About This Book

This book gives you a high-level introduction to pureXML. It's not intended to be a comprehensive or authoritative resource but rather an informal and brief description of the more important aspects of pureXML.

If you want to get into the nitty-gritty of the pureXML features, see the resources listed in Chapter 7. They include two free e-books that describe pureXML in much greater detail.

Regardless of your level of knowledge about XML or your level of technical ability, check out the XML Contest described in Chapter 1. Entering the contest is easy, and you can win prizes. Who knows? You may even discover something about XML and pureXML while winning free stuff!

Foolish Assumptions

Try as we may to be all things to all people, when it comes to writing this booklet, we had to make a few educational guesses about who would be most interested in *DB2 pureXML For Dummies.* Here's what we assumed about you, the reader, as we wrote this book:

- You're a manager or technical person with a basic knowledge of technology. You may even be an IT manager.
- You've heard at least a little bit about DB2 pureXML, IBM's new product.

- You want to use this book to find out the basics about pureXML, but not the gory details.

Whoever you are, welcome.

Conventions Used in This Book

To help you make your way through this book, I use the following conventions:

- *Italic* points out new words and defined terms.
- **Boldface** text highlights keywords in bulleted lists and the action part of numbered steps.
- Text that appears in a gray box without a title indicates example commands and queries. If you want to see the examples in action, enter these commands and queries in the DB2 Command Editor.
- `Monofont` is used for Web addresses.

  When this book was printed, some Web addresses may have needed to break across two lines of text. If that happened, rest assured that there aren't any extra characters (such as hyphens or spaces) to indicate the break. So, when using one of these Web addresses, just type in exactly what you see in this book, pretending as though the line break doesn't exist.

Icons Used in This Book

Throughout the margins in this book, you see little icons that highlight different types of information. We use the following four:

In a hurry? Then don't miss this text because it can save you both time and effort.

If you're interested in the technical aspect of DB2 pureXML, then read this extra information. If not, feel free to skip it.

Don't forget these valuable tidbits.

Be careful. You don't want to fall into these common pitfalls or traps.

Chapter 1

Solving a Storage Problem: DB2 pureXML

In This Chapter

- Examining past approaches to storing data
- Using relational databases to store data
- Taking advantage of IBM's DB2 pureXML capability

Extensible Markup Language (XML) burst onto the scene a decade ago, but robust XML storage solutions took awhile to emerge. Organizations needed a way to not only store information quickly, but also retrieve it. Unfortunately, initial attempts at storing data served merely as workarounds.

Of course, IBM made huge strides in this regard when it added the pureXML capability to DB2. This data server stores your data, whether it be relational data or XML data, and then "serves" that data to the applications that need it.

This chapter charts the evolution of XML storage solutions.

Out with the Past: Storing XML Data the Old Way

In less than ten years, XML has become one of the most common data formats. Many industries now have standards based on XML, allowing organizations to easily share and exchange information.

Of course, XML sprang into action before XML databases arrived onto the scene. As a result, a variety of approaches to storing XML data developed:

- Storing XML data in files
- Stuffing XML data into relational databases
- Shredding XML data into relational databases

Storing XML data in files

One of the first approaches to storing data was to simply store XML data in files. However, people quickly discovered that this approach was unworkable from a performance point-of-view. Here's why: If you store the XML for each record in its own file in a directory and you then need to search for information within a record, you need to open and parse the contents of each file in order to perform the search. This process quickly (or we should say slowly!) makes this approach unworkable.

Also, after people began developing systems based on file system storage, they quickly discovered that they needed common data management features — concurrency, recovery, security, and so on. It wasn't long before people decided to stop reinventing the wheel — after all, databases have had these capabilities for some time — and began using databases.

Stuffing XML data into a relational database

A simple way to store XML data in a relational database is to place the XML data as-is into a single field, a process known as *stuffing*. Figure 1-1 shows the stuffing of XML data into a relational database.

Because it uses a database, this approach benefits from a database's data management capabilities, such as recovery and security. Stuffing also offers increased scalability and improved performance.

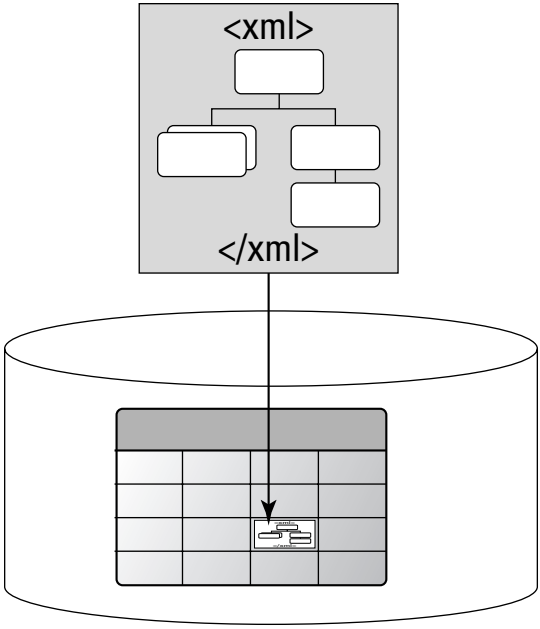

Figure 1-1: Stuffing XML data into a relational database.

However, on the down side, stuffing still has significant performance issues if you need to retrieve or search for information that appears within the XML data.

Shredding XML data into relational databases

The need to improve performance when retrieving a part of the XML data led to the emergence of shredding. *Shredding* involves mapping XML data into relational tables. The various elements, attributes, and data are stored in separate fields. Figure 1-2 shows the shredding of XML data into a relational database.

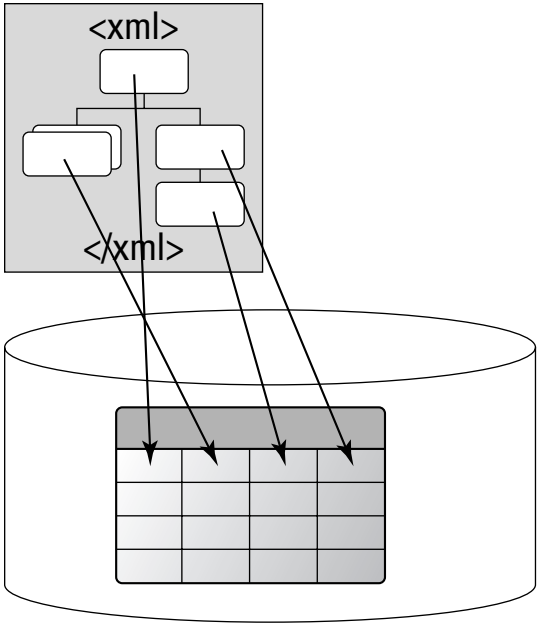

Figure 1-2: Shredding XML data into a relational database.

Of course, the process of decomposing the XML data into the database fields means that placing the XML data *into* the database takes longer. Likewise, the process of recomposing the data from the database to form the original XML means that it takes longer to get the XML data *from* the database.

However, retrieving and searching for individual pieces of information that appear within the XML data is much faster. That's because shredding speeds the retrieval of individual pieces of information at the cost of slowing the insertion of data into the database and slowing the retrieval of the original XML.

Shredding also results in more complex application development and maintenance efforts because of the additional code required to shred the XML data and the overhead of maintaining complex database schemas. For example, if you shred the schema for the FpML industry standard into a relational database, it requires 485 separate tables. Now consider the following:

- The development effort required to map FpML data into those 485 tables
- The maintenance effort required to manage those 485 tables
- The development effort required to reconstitute and join individual pieces of information when retrieving information from the database

In with the New: Native XML Databases

The initial approaches for storing XML were essentially workarounds. (For more on these approaches, see the section "Out with the Past: Storing XML Data the Old Way.") They used existing infrastructure to store XML. What was needed, however, was infrastructure that natively handled XML data — in other words, an infrastructure that directly stored the actual XML data and provided standard ways to work with the XML data.

Fortunately, native XML databases met this need. These XML databases fall into two categories:

- **XML-only databases:** These databases, which store only XML, work well for isolated XML data. In other words, they're suitable when the XML data isn't used with other information. However, using information in isolation is becoming increasingly rare, as organizations try to take full advantage of all information in their environments.

  

  If an organization chooses an XML-only database and later needs to work with both the XML data and other data, then it must engage in a costly integration effort, as well as resulting performance issues from working with more than one non-integrated database.
- **Relational databases that support the native storage of XML data:** This robust solution to XML data management potentially offers improved performance. Now, the ideal solution exists.

DB2 pureXML Arrives

IBM DB2 is a data server. In other words, DB2 can store your data and then "serve" that data to the applications that need it. DB2 works with both relational data and XML data.

DB2 9 introduced the pureXML capability. IBM treats the XML as a collection of XML elements and attributes. It stores the information as actual XML elements and XML attributes, allowing faster data insertion into the database and faster retrieval of information.

This approach is very different than that of other major database vendors who offer different approaches, such as the storage of XML as a string of characters or the storage of XML as a binary object. Of course, when you use other vendors' solutions that don't store XML in a pure format, performance implications result. For example, if you store XML as a string of characters, then you need to parse the characters when working with the XML. If you parse XML every time you read it from the database, you have a significant performance overhead. The only way to optimize performance is to store and work with XML directly.

IBM developed, from the ground up, a pure native XML storage capability and XML engine. The name pureXML is very appropriate because DB2 actually stores and works with the XML in its purest format. It's pure because IBM didn't attempt to retrofit XML into an existing database infrastructure. Instead, it built a new native XML capability that works hand-in-hand with the existing infrastructure. By choosing such an approach, IBM avoids many of the performance hurdles that other vendors encounter.

Figure 1-3 shows the DB2 approach. Notice how the storage is optimized for XML and the storage is separately optimized for relational data. Also notice that common services occur across both types of data, providing language flexibility. In other words, you can use SQL to access the data, or you can use XQuery to retrieve data. (For more on XQuery, see Chapter 4.)

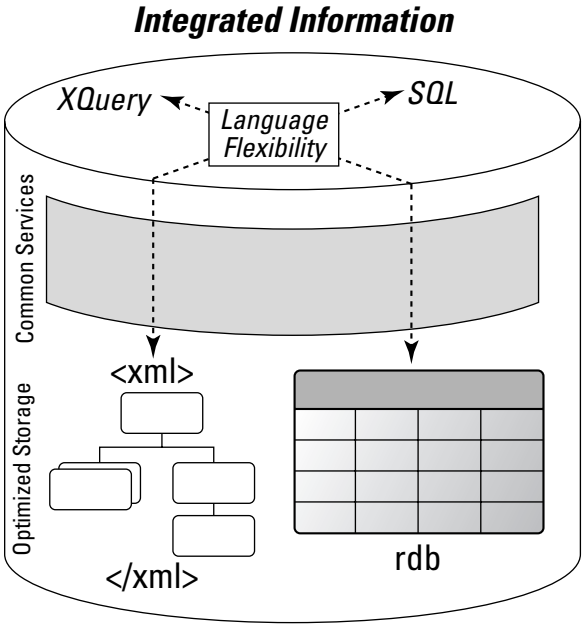

Figure 1-3: IBM DB2 with pureXML.

The ability to store XML data in a database in its purest form provides the following advantages:

- Because you don't need to transform the XML data before inserting it into the database, inserting the XML data into the database is faster. Not only is the performance better, but it requires far less work for database administrators and developers. This is especially important because the costs involved in managing and working with the database are probably the most important costs for you to minimize.
- Because the database natively handles the XML elements and you don't need to parse the XML data when retrieving it from the database, retrieving information from the XML data is faster.

Search for the XML Superstar contest

The International DB2 User Group (IDUG) is running its "Search for the XML Superstar" contest. This contest's goal is to highlight the benefits of DB2 and its related technologies to current and emerging generations of information management professionals — specifically database developers — and university students and faculty.

XML Superstar comes in many different flavors. In fact, the contest is really five contests in one because you don't have to be an expert to participate. Contest winners can range from XML-neophytes to XML-Xperts in one of the five main categories:

- Video Challenge
- Gadget Challenge
- Query Challenge
- Ported Application Challenge
- XML Application Programming Challenge

Points are awarded for participation in each contest. An overall winner, dubbed XML Grand Master, will be selected based on total points gathered during the entire duration of the contest. The XML Grand Master can also be a winner of an individual contest.

The durations and time investment for each contest is different. Contestants can check the status of their entries at any time by logging into the contest Web site at `www.xmlchallenge.com`. To enter any of the contests, participants need to complete the Quick Quiz — a small questionnaire that requires no prior XML or database experience.

The benefits to Contest participants are valuable and plentiful. In addition to honing new skills with high professional demand, participants are eligible to win one of a series of valuable prizes. Students may win opportunities for interviews and lab visits while enhancing the skills that will catch employers' eyes on a resume. Developers can achieve recognition from world-class consultants and be celebrated for standing out among their peers through impressive innovation and technological prowess.

Chapter 2

Getting Started with DB2 pureXML

In This Chapter

- Downloading and installing DB2 Express-C
- Setting up a database table with pureXML

Getting started with pureXML is easy. You can download IBM DB2 for free and start using the pureXML feature immediately after you install it.

In this chapter, you also discover how to create a database, which you can use to insert data, issue queries, and more.

First Things First: Downloading DB2

DB2 comes in many different flavors, and which one you want depends on your planned use. This booklet uses DB2 Express-C. Express-C, a fully functional version that is completely free to download, develop, deploy, test, run, embed, and redistribute.

You can run DB2 Express-C in production environments, and you have no data storage limits. Your only restrictions are that you run DB2 Express-C on a server with up to 2 CPUs and up to 4GB of memory.

DB2 Express-C is available for Linux, Unix, and Windows running 32-bit or 64-bit. You can download DB2 Express-C from the following Web page:

```
www.ibm.com/software/data/db2/express
```

Using the Setup Wizard to Install DB2

You can install DB2 in several ways, although one of the easiest is to use the Setup wizard. (For information about using the other ways to install DB2, see the DB2 documentation.)

At each step in the Setup wizard, you can click Help to get instructions for that step. ***Note:*** The installation procedure is subject to minor changes as feedback comes in from users. Don't be alarmed if the screens don't exactly match what you see in this section. You can typically accept the default settings, and remember that help is always available.

To use the setup wizard to install DB2:

1. **Execute setup.exe.**

   The Welcome screen appears and provides you with a number of options.

2. **Choose Install a Product.**

3. **From the list, choose DB2 Express-C as the software that you want to install.**

4. **When the Setup Wizard appears, click Next.**

Getting additional help

Downloading and installing DB2 Express-C is straightforward, and each screen in the installation includes a Help button. However, if you do run into any problems or if you have any questions, you can get answers in the following online forum:

```
www.ibm.com/developerworks/forums/dw_forum.jsp?
   forum=805&cat=19
```

5. **Read the license agreement and then select the checkbox to accept its terms; click Next.**

   You have your choice of several installation types: Typical, Compact, and Custom.

6. **Choose Typical and click Next.**

   You are now asked if you want to create a response file.

7. **Choose Install DB2 Universal Database Express Edition on this computer and click Next.**

8. **Choose the default language that you'd like DB2 to use and click Next.**

   You're asked to specify the location where DB2 products are installed on your computer (see Figure 2-1).

9. **Choose the default value of DB2 and click Next.**

10. **When asked to specify the location of the DB2 Information Center, choose On the IBM Web Site and click Next.**

    The next screen, shown in Figure 2-2, asks for the user information used to start DB2 Administration server.

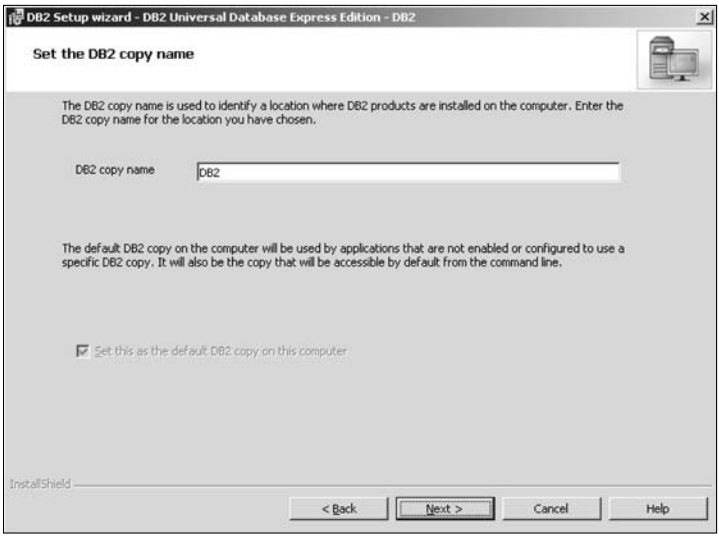

Figure 2-1: Setting the DB2 copy name.

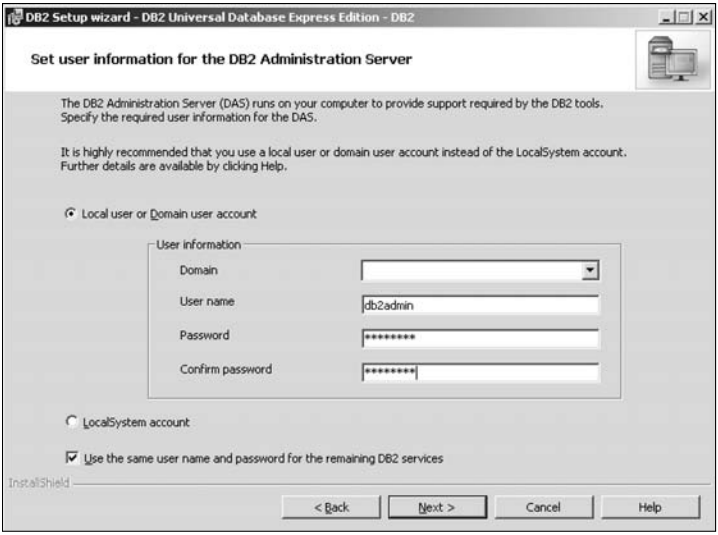

Figure 2-2: Setting user information.

11. **Choose Local user or Domain user account.**
12. **Choose a user name and a password.**

    Make sure to write down the user name and password in a place that you'll remember to check. Nothing is more frustrating than evaluating software and then returning to it weeks later and not remembering the user name and password.

13. **Select the checkbox for using the same user name and password for the remaining DB2 services and click Next.**

    The next screen asks you to configure the DB2 instances that you want to create as part of the installation

14. **Click Next to select the default instance.**
15. **Click Next to select the default tools catalog options.**

16. **Uncheck the option to set up DB2 send notifications automatically to the SMTP server when a database needs attention and click Next.**

    The security settings, shown in Figure 2-3, appear.

17. **Click Next.**

    This setting ensures that only users who belong to DB2 users group (DB2USERS) and DB2 administrator group (DBADMINS) can access DB2 objects.

    Now the Setup wizard copies the files.

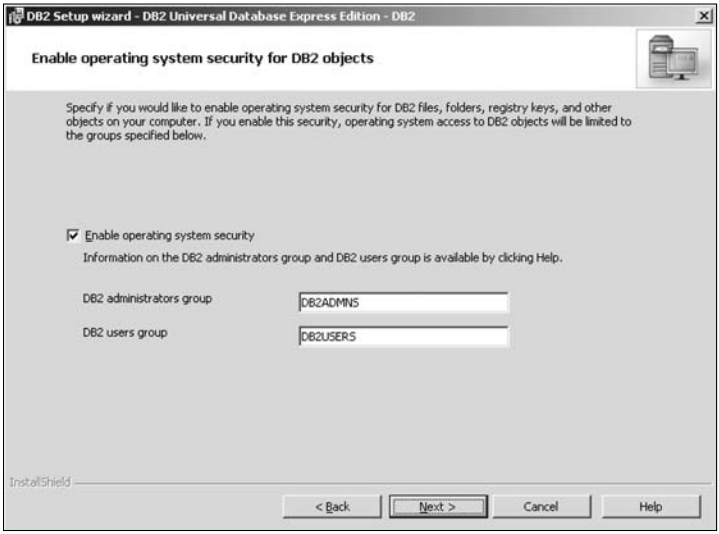

Figure 2-3: Enabling operating system security.

Creating Your First Database Table

After you download and install DB2 Express-C, you're ready to create your first database with pureXML.

pureXML is simply the ability to store XML in the database in its purest form. All that's involved in setting up a database table for pureXML is to indicate that an area of the database will contain XML data. After you do that, you receive all the benefits of pureXML: You can more easily set up the storing and retrieval of information, and, just as important, you'll enjoy faster storage and retrieval of XML.

IBM DB2 allows you to create a table containing only XML data. You can also to create a table with a mixture of relational and XML data. In addition, a row in a database table can have multiple XML columns.

Although you can create a database table in many ways, we use the Command Editor in the DB2 Control Center. (For information about using the other ways to create a database table, refer to the DB2 documentation.)

To create a database with pureXML:

1. **Start the Control Center.**

   From the Windows Start menu, select Start➪All Programs➪IBM DB2➪DB2COPY1 (Default)➪General Administration Tools➪Control Center, as shown in Figure 2-4.

   The Control Center window appears.

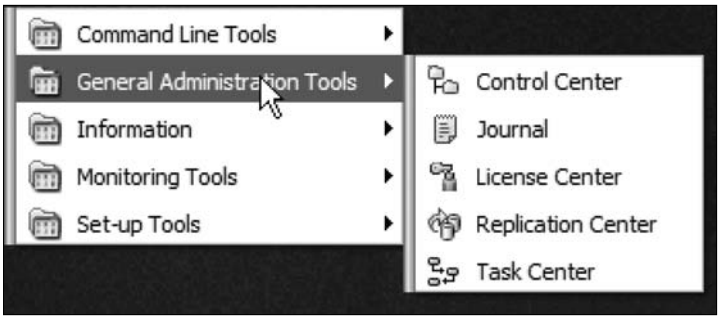

Figure 2-4: Starting the IBM DB2 Control Center to create a database.

2. **Launch the Command Editor.**

   In the Control Center toolbar, click the Command Editor icon, shown in Figure 2-5.

   The Command Editor window appears.

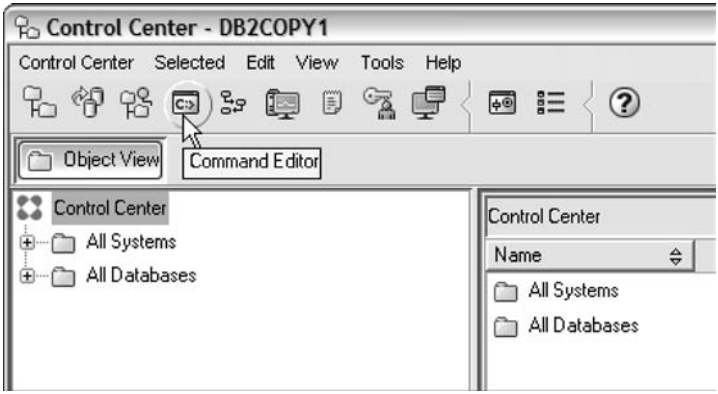

Figure 2-5: Launching the Command Editor.

3. **Create the database.**

   In the upper pane, enter the following command and then click the green Execute icon:

```
CREATE DATABASE dummies USING CODESET
utf-8 TERRITORY US
```

   By default, DB2 uses the UTF-8 codeset, which is Unicode compliant.

   Figure 2-6 shows where to enter the command and the location of the green Execute icon.

4. **Connect to the database.**

   Clear the contents of the top pane by selecting the command that created the database and pressing the Delete key. In the upper pane, enter the following command and then click the green Execute icon:

```
connect to dummies
```

5. **After a connection to the dummies database is established, create a database table.**

   Clear the contents of the top pane by selecting the command that created the database and pressing the Delete key. In the upper pane, enter the following command and then click the green Execute icon (see Figure 2-7):

```
create table orders (
id int primary key generated always as
identity,
info xml
)
```

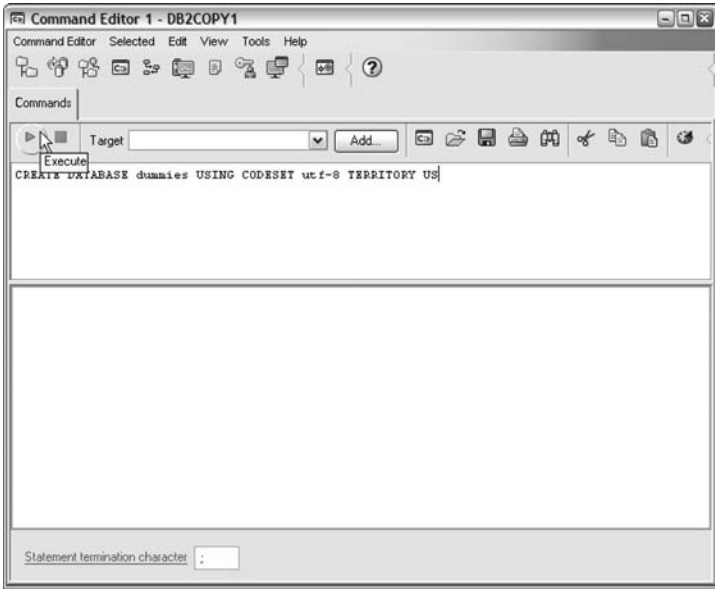

Figure 2-6: Creating the database.

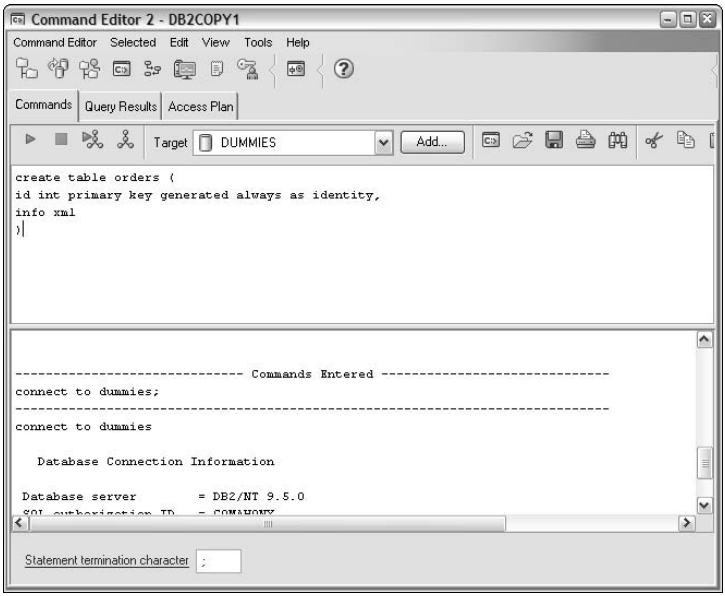

Figure 2-7: Creating the database table.

You've now created a database table with two columns: one column contains a unique identifier that is the primary key for the database, and another column that contains XML data.

Chapter 3

Working with XML Data

In This Chapter

- Adding information to your database
- Displaying your table
- Placing data into your database
- Making changes to data

After you create a database with a table that stores XML data, you're ready to insert data into that database. As your information changes — customers move, for example — you can edit your data as necessary. (For information on creating a database, see Chapter 2.)

Inserting XML Data into the Database

You can use the `INSERT` SQL statement to insert XML data into the database. (If you haven't created a database, see Chapter 2.) Use the `INSERT` statement in the following circumstances:

- Whenever you want to interactively add data to the database, as database administrators sometimes do.
- When you're writing programming code for a software application and have the XML data in a variable.

You can issue the INSERT SQL statement in several ways:

- **You can enter it in the Command Editor of the DB2 Control Center:** This method is how you typically issue the INSERT statement when you're learning DB2 or if you're a database administrator.
- **You can include it in the code you're writing for a software application that works with DB2.** In such scenarios, you typically use the INSERT statement when you're writing an application, and you have the XML data available in a variable.

This booklet doesn't describe the syntax and usage of the INSERT SQL statement. Many resources can provide more information on SQL statements, including the DB2 documentation and various Web sites.

To insert data into a database:

1. **Start the Control Center and launch the Command Editor.**
2. **Connect to the database.**

   In the upper pane, enter the following command and then click the green Execute icon:

   ```
   connect to dummies
   ```

3. **Insert data.**

   Clear the contents of the top pane by selecting the command that connected to the database and pressing the Delete key. In the upper pane, enter the following command and then click the green Execute icon:

   ```
   insert into orders (info) values
   ('<order><name>John Smith</name><addr>
   <street>111MainSt.</street><city>Dallas
   </city><state>TX</state><zip>00112
   </zip></addr><book>pureXML ForDummies
   </book><quantity>1</quantity><price>
   27.99</price></order>')
   ```

4. **Insert another record.**

   Clear the contents of the top pane. In the upper pane, enter the following command and then click the green Execute icon:

```
insert into orders (info) values
('<order><name>Jane Doe</name><addr>
<street>99 High Street</street><city>
Cambridge</city><state>MA</state><zip>
02139</zip></addr><book>pureXML For
Dummies</book><quantity>2</quantity>
<price>27.99</price ></order>')
```

If you're using the database created in Chapter 2, the id column is automatically generated by DB2 each time you insert data into the database because that is how you originally defined the database. Therefore, to populate a row of the database, you need to insert data only into the `info` column.

These `INSERT` statements insert a row into the orders table:

Table 3-1 How the INSERT Statement Works

id	*info*
1	<order> <name>John Smith</name> <addr> <street>111 Main St.</street> <city>Dallas</city> <state>TX</state> <zip>00112</zip> </addr> <book>pureXML For Dummies</book> <quantity>1</quantity> <price>27.99</price> </order>

(continued)

Table 3-1 ***(continued)***

id	*info*
2	<order> <name>Jane Doe</name> <addr> <street>99 High Street</street> <city>Cambridge</city> <state>MA</state> <zip>02139</zip> </addr> <book>pureXML For Dummies</book> <quantity>2</quantity> <price>27.99</price> </order>

If the XML you're inserting is large or complex, typing the XML data into the `INSERT` statement is impractical. In most cases, you'd write an application to insert the data using a host variable or a parameter marker.

Viewing Your Data

If you want to see the data in your database:

1. **On the left-hand navigation pane of the Control Center window, double-click All Databases.**
2. **Double-click the DUMMIES database.**
3. **Click the Tables folder.**
4. **Double-click the ORDERS table.**

   A window displays the table's contents (see Figure 3-1).

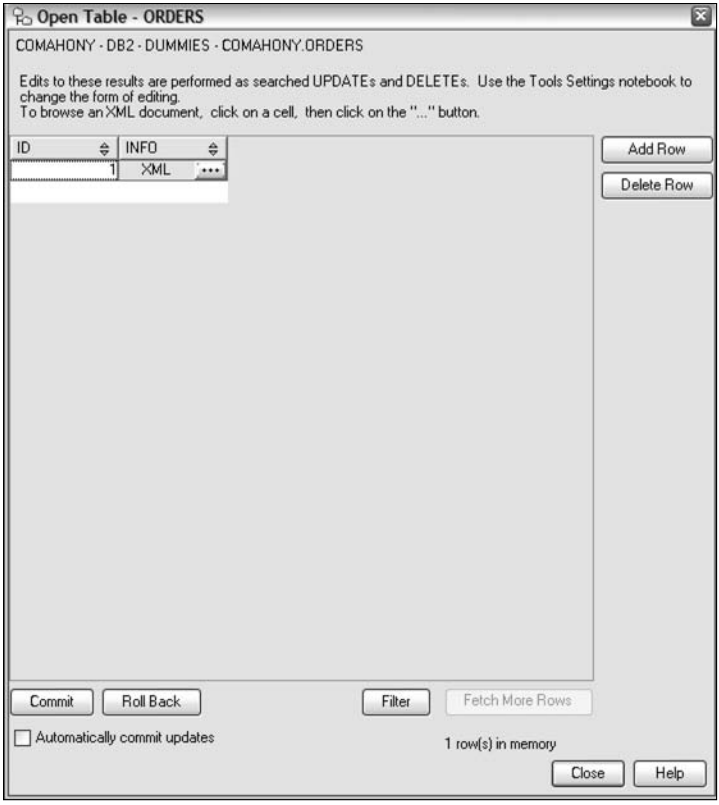

Figure 3-1: Contents of the database table after inserting data.

5. **To see the XML, click ... in the INFO column.**

   A window shows you the XML data (see Figure 3-2).

   This window lets you examine the XML data using a tree view or by viewing the XML source code.

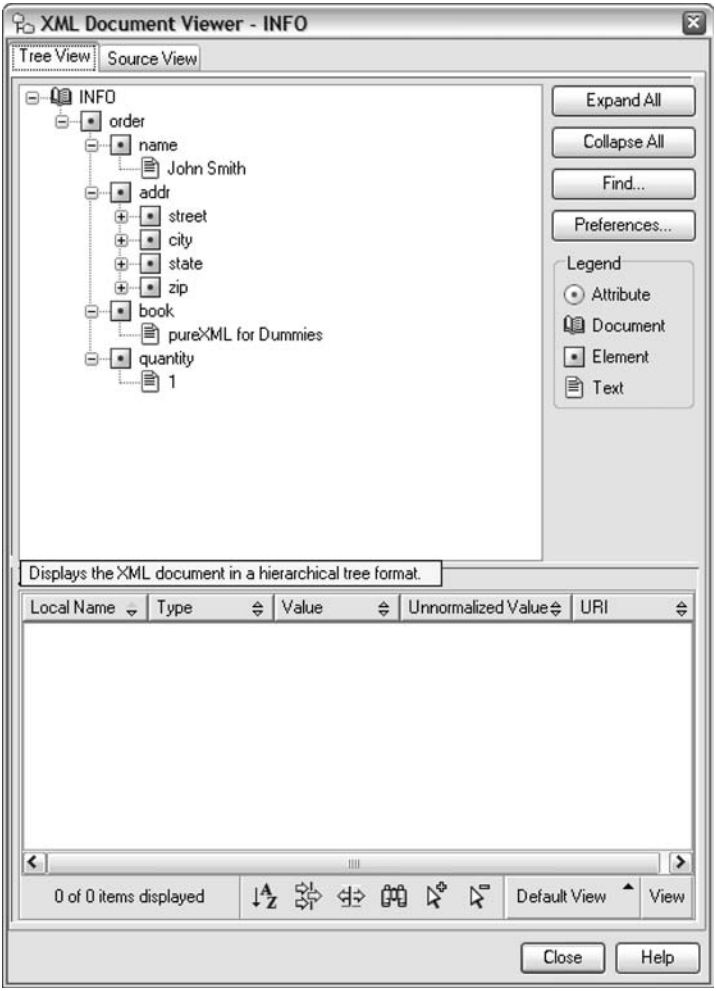

Figure 3-2: The windows showing the XML data.

Importing Data into Your Database

DB2 has a special `IMPORT` command that allows you to import data — including XML data — that resides in files into the database. The `IMPORT` facility loads the data from a

delimited ASCII file. Each line in the ASCII file represents a row of data in the database. In each line, a delimiter separates the data for each column.

The usual convention is to use the `.del` file extension for ASCII delimited files.

You can use either of the following ways to import XML data:

- Enclose the XML data in single quotes to include it directly in the ASCII file that is imported.
- Include a reference to an external file that contains the XML data.

In the latter method, each XML file contains the XML data for a single database cell. To include a reference to an external file that contains the XML data, you use the XML Data Specifier (XDS) syntax. (For more information about the XDS syntax, see the DB2 documentation.)

The following sample `IMPORT` command imports XML data from `C:\orders.del` into the `orders` table in the dummies database:

```
IMPORT FROM "C:\orders.del" OF DEL XML FROM
"C:\orders" INSERT INTO dummies.orders;
```

If you have large amounts of XML data in a file, you may want to use the `LOAD` command, which has better performance for large amounts of data. This sample `LOAD` command follows. It loads XML data from `C:\orders.del` into the `orders` table in the dummies database:

```
LOAD FROM "C:\orders.del" OF DEL INSERT INTO
dummies.orders;
```

Updating XML Data

Sometimes you need to update information in a database. For example, you may need to update a customer address when the customer moves. Fortunately, updating XML data is efficient. Because it stores XML data in a pure native form, DB2 performs the updates without needing to first parse the XML data.

A database column consists of individual database cells. For each database row, you see one cell in the column. To replace all the XML data in a database cell, simply use the SQL `UPDATE` command.

The following example works with the database created in Chapter 2. In this example, imagine that the customer with `id` set to `1` has moved to a new house. The following command, typed in the Command Editor of the DB2 Control Center, replaces the XML data in the `info` column of the row with its `id` set to `1`:

```
UPDATE orders SET info = XMLPARSE ( DOCUMENT
('<order><name>John
Smith</name><addr><street>123
Broadway</street><city>New
York</city><state>NY</state><zip>10003</zip><
/addr><book>pureXML For
Dummies</book><quantity>1</quantity><price>27
.99</price></order>'))WHERE id = 1
```

To check that the XML is updated, in the Control Center window, double-click All Databases, double-click DUMMIES, click the Tables folder, and finally double-click the ORDERS table.

Updating only a specific part of XML data

You can use XQuery to rename, insert, delete, replace, or change the value of individual elements or attributes in an XML document. You can also update XML data in many ways — way too many to even summarize in this booklet. Instead, read this great article on the Web, which not only uses examples to describe the different ways that you can update the XML data, but includes information about common pitfalls and their workarounds:

`www.ibm.com/developerworks/db2/library/techarticle/dm-0710nicola/`

Chapter 4

Retrieving XML Data from Your Database

In This Chapter

- Getting the scoop on query types
- Using XQuery to retrieve files
- Knowing when to use each query language
- Specifying paths with XPath
- Creating indexes to speed up the retrieval process

What good is information if you can't work with it? Fortunately, pureXML makes retrieving your data a simple process through queries.

In this chapter, you discover how you can use queries to quickly find the information that you need. In addition, you see how indexes can speed up the retrieval of certain information.

Issuing Queries to Find Data

Maybe you want to create a software application that allows users to add, remove, and update the information in the database. Or perhaps you want to create an application that generates reports about the information in the database. Either way, you're in luck. You can update and retrieve information in your database by using something called a *query*. You use special languages to issue queries to the database and when you do, the database either updates or retrieves the information in the database.

The most well-known query language is SQL (Structured Query Language). SQL allows users to update or retrieve information in a relational database. If you want to issue queries against the individual pieces of information in the XML data, you can use XML extensions to SQL or a new query language called *XQuery*.

This distinction between the retrieving the XML data as a whole and retrieving individual elements in the XML data is important. After all, if you're working with the XML data as a whole, then relational large object storage is sufficient, and all you need is traditional SQL. However, if you want to work with the individual pieces of information in the XML data, then you'll likely want to use an XML column and the query languages for working with an XML column data.

DB2 provides two ways to issue queries for information in XML data:

- XQuery
- SQL/XML

You can actually issue queries for XML in many ways, including: traditional SQL, SQL/XML, XQuery, and XQuery with embedded SQL. However, this chapter concentrates on the preceding two methods.

Xceptional retrieving: XQuery

XQuery is a query language for working with XML data. XQuery offers developers a strong combination of programming power and ease of use.

This industry standard is actually a *W3C Recommendation* — a standard recommended by the World Wide Web Consortium (W3C), which is the main international standards organization for the World Wide Web.

XQuery has many of the features of a programming language like variables, data types, operators, conditional expressions, functions, and so on. Two of the most important types of expression in XQuery are

- FLWOR expressions
- XPath expressions

The following sections provide a brief overview of these expressions.

FLWOR expressions

FLWOR (pronounced "flower") expressions get their name from the first letters of the major parts of the expression:

- `for`
- `let`
- `where`
- `order by`
- `return`

If you're familiar with SQL, a FLWOR expression is similar to its SELECT-FROM-WHERE expression.

FLWOR expressions *iterate* over a number of clauses. This is geek speak for saying that XQuery executes the sequence of clauses in a FLWOR expression for each piece of data provided to it. The following list looks at each *clause* — the technical term for each part of the expression — in more detail:

- `for` specifies the iteration sequence. In other words, this clause specifies how many times XQuery executes the sequence of clauses. A variable indicates the current iteration.
- `let` declares a variable and assigns a value to the variable. You can assign any value to this variable, including a list containing multiple items.
- `where` specifies criteria for filtering the information that is returned.
- `order` by specifies the order of the information that is returned.
- `return` specifies the information that is returned.

To execute an XQuery directly in DB2, preface the query with the keyword `xquery`. If you embed XQuery expressions in SQL, you don't need to preface them with `xquery`.

The following FLWOR expression, entered in the Command Editor of the DB2 Control Center, works on the example database used in earlier chapters of this booklet:

```
xquery
for $y in db2-
fn:xmlcolumn('ORDERS.INFO')/order
where $y/addr/zip="02139"
return $y
```

The db2-fn:xmlcolumn is a DB2 convention for identifying the DB2 data, rather than part of the XQuery standard.

This expression iterates through each row that has an order node in the `ORDERS.INFO` column. If a node has its ZIP code set to `02139`, it returns the XML data.

XQuery (and XPath) expressions are case-sensitive. If you reference the `Order` element in your XQuery expression, it won't match the `order` node in the XML.

XPath expressions

XPath allows you to identify information in XML data. For example, you can specify that you want to see the names for all orders where the city is "Dallas" and the price is greater than $35.

XPath expressions let you specify the path to the XML data you want. They also allow you to specify conditions that must be met in order for the information to be returned.

XPath is a very powerful language. In fact, the last example in the preceding section includes an XPath expression: `$y/addr/zip="02139"`. It identifies that you want XML elements where a `ZIP` element that is within an `addr` element has the value `02139`.

We cover XPath in more detail in the section "Identifying Information with XPath," later in this chapter.

Extending with SQL/XML

SQL/XML is a set of extensions to SQL for working with XML data. It consists of adding the following features, and others, to SQL:

- The XML data type
- The ability to use XPath and XQuery to search for specific information in XML data
- Functions that allow you to publish relational data in XML format
- Functions that allow you to convert relational data to XML data, and vice versa
- Functions to validate XML data against XML schemas

SQL/XML is also an industry standard — it's an ISO and ANSI standard — developed by INCITS H2.3, with participation from Oracle, IBM, Microsoft, and others.

The following SQL/XML expression works on the database created in earlier chapters of this book. It iterates through each row that has an `order` node in the `ORDERS.INFO` column. If a node has its ZIP code set to 02139, it returns the value of the corresponding order node:

```
Select info from orders
Where xmlexists('$c/order/addr[zip="02139"]'
passing orders.info as "c")
```

Deciding Which Query Language to Use

Developers comfortable with XML development will easily adapt to XQuery and should consider using XQuery together with SQL. (SQL is necessary for certain operations, such as inserting, updating, and deleting data on a database cell level.)

SQL/XML, on the other hand, is ideal for environments where developers aren't as comfortable with XML development. SQL/XML consists of extensions to the SQL language, and SQL is a quite mature language. SQL programming has strong support from the development community, both from a skills perspective and from a tool support perspective.

DB2 uses the same compiler to execute queries regardless of the query language that you use. Thus, performance differences between the two languages are minimized.

Table 4-1 compares the ability of XQuery and SQL/XML to perform routine operations in IBM DB2.

Table 4-1 Comparing XQuery and SQL/XML

Operation	*XQuery*	*SQL/XML*	*Comments*
Inserting an XML document	No	Yes	You use SQL to insert an entire document.
Retrieving an XML document	Yes	Yes	
Retrieving part of an XML document	Yes	Yes	
Using predicates with relational data	Yes	Yes	XQuery doesn't support relational predicates. However, IBM DB2 supports SQL in XQuery, allowing predicates with relational data.
Deleting an XML document	No	Yes	You use SQL to delete an entire document.
Updating an XML document	Yes	Yes	
Updating part of an XML document	Yes	Yes	
Joining XML data	Yes	Yes	Using XQuery is the easier approach. Using SQL/XML is typically difficult to code.

Operation	*XQuery*	*SQL/XML*	*Comments*
Joining XML with relational data	Yes	Yes	XQuery doesn't support joins to relational data. However, IBM DB2 supports SQL in XQuery, allowing joins to relational data.
Transforming XML	Yes	Yes	Using XQuery is the easier approach. Using SQL/XML is typically difficult to code.
Aggregating XML data	Yes	Yes	Using SQL/XML is the easier approach. Using XQuery is possible with embedded SQL, but is typically difficult to code.
Calling external functions	No	Yes	
Passing parameter markers	No	Yes	

At first glance, it may appear that SQL/XML has more extensive support. However, please note the comments indicating that some operations are far easier with XQuery. This ease of coding can make a significant difference for projects.

Identifying Information with XPath

SQL/XML and, it turn, XQuery use XPath to identify individual pieces of information in XML data. This explanation is an oversimplification, because you can't use XPath to point to just any information in the XML data.

Visualize XML as a hierarchical tree of nodes. You can use XPath to retrieve information from one or more of those nodes.

Like XQuery, XPath is also a W3C Recommendation.

To choose XML nodes, XPath uses expressions that include

- The path to nodes
- Filters (optional)
- Operators (optional)

Entire books are dedicated to XPath. This booklet includes a high-level summary of XPath.

The path to nodes

To specify the path to the nodes, you use a special syntax that uses the special characters shown in Table 4-2.

Table 4-2 Specifying the Path to Nodes

Expression	*Description*
`/node_name`	Select the nodes called `node_name` that appear directly under the current node
`//node_name`	Select the nodes called `node_name` that appear anywhere under the current node
`.`	Select the current node
`..`	Select the parent of the current node
`@attribute_name`	Select the attribute node with the name `attribute_name`

Note: You need to plug in the actual node and attribute names that you want to specify in place of *node_name* and *attribute_name* in Table 4-2.

In these node selection expressions, you can optionally use special character sequences called wildcard characters (see Table 4-3). These wildcard characters replace *node_name* and *attribute_name* in the expressions. They allow you to specify that you want all nodes, and not just the nodes with a specific name.

Table 4-3 XPath Wildcards	
Wildcard	***Description***
`*`	Matches all element nodes
`@*`	Matches all attribute nodes
`node()`	Matches all nodes. (More kinds of nodes exist than just element and attribute nodes.)

Table 4-4 includes examples of node selection expressions.

Table 4-4 Node Selection Expressions	
Expression	***What It Does***
`/order/name`	Selects all `name` nodes that are children of `order`
`/order/addr/*`	Selects all children nodes of the `addr` nodes that appears under `order`
`//name`	Selects all `name` nodes no matter where they are in the XML data
`/order//city`	Selects all `city` nodes that are descendant of the `order` node, no matter where they are under the `order` node

Filters

XPath allows you to filter the nodes. For example, you can specify that you want to choose only orders where the city is Dallas. In geek speak, you're specifying predicates.

Predicates are enclosed in square brackets. Table 4-5 lists some common predicate expressions.

Table 4-5 Common Predicate Expressions

Predicate Expression	*What It Does*
`/order/book[1]`	Selects the first `book` node that is a child of order
`/order/book[last()]`	Selects the last `book` node that is a child of `order`

Operators

XPath supports a wide range of operators, including the ones listed in Table 4-6.

Table 4-6 XPath Operators

Operator	*Description*
`|`	Tells XPath to compute two paths and use values from both paths.
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`div`	Division
`=`	Equal
`!=`	Not equal
`<`	Less than
`>`	Greater than
`or`	Or
`and`	And

Table 4-7 shows examples of expressions with operators.

Table 4-7 XPath Wildcards

Expression	*What It Does*
`/order/book[last()-1]`	Selects the second last `book` node that is a child of `order`
`/order/book[quantity>1]`	Selects all `book` nodes that are children of `order`, where the value of `quantity` is greater than 1
`//order/name | //order/addr/zip`	Selects all the `name` nodes and all the `zip` nodes.

Using Indexes to Improve Performance

Sometimes you need to speed up the retrieval of information in a relational database. Perhaps you have an application that allows people to search for orders by specifying the name of the person who placed the order. However, when someone searches for an account, it seems to take forever. In such situations, it doesn't usually take long for people to complain about the spinning hour glass, and before you know it, you're asked to speed up the display of the account information.

In such situations, you can build an index that speeds the retrieval of account information. The index is like a table with two columns: One column contains the information that you use when searching, and the other column contains a pointer to the account information in the database.

In the preceding example, the first column would contain the name of the account holder, which is what you're using to search for the account information. Of course, in reality, indexes are more complicated than this description, but the basic idea is the same.

Using an index is faster than searching the database table for two reasons:

- Because index rows are smaller in size than corresponding database table rows, you can fit more of them in a physical file block. This benefit is important because it means that when you read each physical file block, you can search a much greater number of records. Therefore, by reducing the number of physical file blocks that you need to read, you speed up the retrieval of information.
- Indexes order items sequentially based upon the information you're searching against. This sequential approach allows for the use of some very fast search techniques.

Information that doesn't change often but is queried frequently is a good candidate for indexing. DB2 9 includes tools that help you figure out whether data is a good candidate for indexing.

While using an index speeds up the retrieval of information, it does so at a cost. Just for starters, you must store, maintain, and back up the index so you have additional storage and maintenance costs. You also have performance costs associated with the upkeep of the index. Every time a database insertion, update, or deletion occurs, the index will need to be updated, which in turn adds to the workload of the database server.

Comparing relational and XML indexes

DB2 allows you to create an index for XML data. However, a traditional relational index and an XML index do have some differences. For example, when you create a traditional relational index, you specify the columns that you want to index. However, when you create an index for XML data, you specify

- The column containing the XML data
- An XML pattern, which specifies the information in the XML data that you want to include in the index
- The type of data you're indexing

As a result, an XML index is different than a relational index in the following ways:

- You can specify only one column in an XML index. Typical relational implementations allow you to include multiple columns.
- You can have zero, one, or multiple XML nodes in each cell of the database column matching the specified XML pattern. Therefore, you can have multiple index entries for a single row in a table.
- You can have multiple indexes for a single XML column, with each index related to a different portion of the XML.

Creating an index

To create an index for pureXML data, you use the familiar `CREATE INDEX` statement. However, you make a few changes to the arguments. In addition to specifying the column to index, you also specify an XML pattern that identifies the information within the XML data that you want to index, and you specify the data type of that information. For example, look at the following sample `CREATE INDEX` statement:

```
CREATE INDEX odindex ON orders(info) GENERATE
      KEY USING XMLPATTERN '/order/orderdate'
as SQL DATE;
```

This statement creates an index called odindex on the info column in the `orders` table. In the index, it includes the information from `orderdate` XML elements that occur within `order` XML elements. And it stores this information using the `SQL DATE` data type.

Keep the following pointers in mind:

- You can index XML data only from a single column. The current version of DB2 doesn't support composite XML indexes.
- If an XML node matches the XML pattern, but the information in that node doesn't match the specified data type, then no index entry is created for that node and an error isn't generated. To configure DB2 to generate errors, make sure to set `REJECT INVALID VALUES` for your table.
- To optimize the performance for your index, ensure that your index is as lean as possible by making your XML pattern as precise as possible. In particular, where possible, avoid using `*` and `//` when defining XML indexes.

Chapter 5

Working with XML Schemas

In This Chapter

- Discovering XML data schemas
- Signing up for validation
- Double-checking your data
- Automatically validating XML data
- Keeping your schema current

If you want to get a handle on the structure of your XML data, you need to look at its schema. You'll also want to verify that your data is adhering to its schema so that you don't encounter problems down the road. This chapter provides an overview of the XML schema support in DB2.

The Schema of Things in DB2

An XML *schema* defines the structure of XML data. For example, an XML schema not only describes the XML elements and attributes that appear in XML data, but where they appear as well. You can consider an XML schema to be a set of rules for the XML elements and attributes that appear in your XML data.

Unlike some other database products, DB2 doesn't force you to *validate* — check that the XML data adheres to the rules of an XML schema — XML data when entering it into the database. The only thing that DB2 does is make sure that the XML

data is well-formed. Well-formed data has the following characteristics, as well as others:

- It has a single root.
- Each XML tag has a matching end tag.
- Each XML tag is properly nested.
- XML tags match case.
- Attribute values are enclosed in quotes.

When inserting XML data into the database, you have the option of specifying that you want to validate the XML data against an XML schema. However, you can also choose not to validate the XML data at this time. This feature has one important implication: It means that the XML data can be validated against different XML schema. Because the XML schema isn't fixed for a database column, you can validate the different cells in a column against different XML schema.

This flexibility can be very beneficial when an XML schema changes, especially when the schema changes aren't backwards-compatible. Instead of possibly having to create new columns and migrate the old XML data, you simply validate the new data against the new schema and validate the old data against the old schema.

Some database vendors restrict you to one XML schema for a database column. Obviously, this restriction creates problems for users of those database systems when schema updates occur.

Registering an XML Schema

If you want to validate XML data in the database, you need to register the XML schema for your database and instruct DB2 to validate the XML data.

If your XML schema is in a single file that doesn't reference other schemas, you need issue only one command to register it. Otherwise, you need to issue an individual command to register the primary XML schema, individual commands to register each of the other schemas, and a command to complete the registration process.

This sample command registers a schema that is in a single file:

```
register xmlschema 'http://mysample.org' from
'C:/orders.xsd' as dummies.orders complete;
```

In this example,

- `http://mysample.org` is the namespace.
- `C:/orders.xsd` is the path to the XML schema file.
- `dummies.orders` is how SQL refers to the schema.
- `complete` tells DB2 to complete the registration process.

XML files often include a pointer to the location of their XML schema files, which are accessible by URI (Uniform Resource Identifier). However, by registering the schema files in DB2, you allow DB2 to work with local copies of the schema instead of incurring the performance overhead of retrieving external files.

Validating XML Data

Validating XML data is a good idea in many environments because it ensures that you won't later encounter issues when working with the data.

Once the XML schema is registered, you can configure DB2 to validate XML data against the schema. For example, you can specify that you want to validate the XML data during `INSERT`, `LOAD`, or `IMPORT` operations by simply using the `XMLVALIDATE` clause within the scope of your `INSERT`, `LOAD`, or `IMPORT` command.

The following sample command validates XML data during a database `INSERT`:

```
insert into orders (info) values (
xmlvalidate('<order><name>Jane
Doe</name><addr><street>99 High
Street</street><city>Cambridge</city><state>M
A</state><zip>02139</zip></addr><book>pureXML
for
Dummies</book><quantity>2</quantity><price>27
.99</price ></order>' according to xmlschema
id dummies.orders))
```

DTDs and validations

DB2 supports DTDs (Document Type Definitions) and external entries for entity resolution, but not for validation. To validate XML documents, DB2 supports only XML schemas. Note that you can convert DTDs to XML schemas without any information loss.

If DB2 determines that the XML data doesn't adhere to the schema, it won't insert that row in the database.

A 1-2 Punch: Check Constraint and Before Trigger

You can use a *check constraint* to ensure that only validated XML data is entered into a database column. When you define a check constraint on a database column, DB2 checks to make sure that the constraint is valid before inserting data into the column. You can use the `IS VALIDATED` and `IS NOT VALIDATED ACCORDING TO XMLSCHEMA` clauses to make sure that only valid XML data is entered into the database column.

A check constraint can't actually validate the XML data. This validation must be done elsewhere — for example, as part of an `INSERT`, `LOAD`, or `IMPORT` command.

A *before trigger*, on the other hand, automatically validates XML data before an insert, an update, or a delete operation. You can specify the `XMLVALIDATE` function in a before trigger instead of in your `INSERT`, `LOAD`, or `IMPORT` commands. (You can find more information about triggers in the DB2 documentation.)

By using both a check constraint and a before trigger, you can ensure that XML data is valid. The before trigger automatically validates the XML data, while the check constraint ensures that the XML data was validated. This approach is failsafe.

Updating an XML Schema

You can use the `XSR_UPDATE` stored procedure to update an XML schema to a new version, provided the new schema is compatible with the old schema. In other words, existing documents in the database that were validated with the old schema must also validate with the new schema.

To update the schema, register the new schema and then call the `XSR_UPDATE` stored procedure. When using this stored procedure, you can choose to retain the old schema or delete it.

Chapter 6

Increasing Your Productivity and Application Performance

In This Chapter

- Determining granularity
- Creating efficient XPath expressions
- Developing lean indexes for XML
- Discovering how RUNSTATS can help
- Optimizing short and repetitive queries

After you master the basics of working with DB2 pureXML, you're ready to take it to the next level. The tips in this chapter not only make you more productive, but also improve the performance of your applications.

These tips were taken from an article by Matthias Nicola titled "15 best practices for pureXML performance in DB2." You can read the original article for additional tips, as well as more details on the tips given in this chapter. You can find the article at `www.ibm.com/developerworks/db2/library/techarticle/dm-0610nicola`.

Choose Your XML Document Granularity Wisely

When you design your XML application and especially your XML document structure, you may have a choice to define which business data is kept together in a single XML document.

Granularity refers to the amount of data in your documents. Check out Table 6-1. There are three levels are granularity:

- **Medium granularity:** Using one XML document per department is called medium granularity, which is a reasonable choice if the department is the predominant granularity at which the application accesses and processes the data.
- **Coarse granularity:** Alternatively, you could combine multiple or many departments into a single XML document — for example, to group all those departments that belong to one unit, which is called coarse granularity. This, however, is suboptimal if you typically process only one department at a time.
- **Fine granularity:** You can also have one XML document per employee, which is called fine granularity, with an additional dept attribute for each employee to indicate which department he belongs to.

  This option is a good choice if employees are often accessed and processed independently from the other employees in the same department. However, if the application typically processes all employees in one department together, one XML document per department can be better.

Table 6-1 Determining Granularity

unitID	*deptdoc*
WWPR	<dept deptID='PR27'> <employee id='901'> <name>Jim Qu</name> <phone>408 555 1212</phone> </employee> <employee id='902'> <name>Peter Pan</name> <office>216</office> </employee> </dept>
WWPR	<dept deptID='V15'> <employee id='673'> <name>Matt Foreman</name> <phone>416 891 7301</phone> <office>216</office> </employee> <description>This dept supports sales world wide</description> </dept>

We don't recommend batching up many independent business objects in a single document. DB2 uses indexes over XML data to filter on a per-document level. (For more on indexes, see Chapter 4.) Therefore, the finer your XML document granularity, the higher your potential benefit from index-based access. Also, if your application uses a DOM parser to ingest the XML retrieved from DB2, small documents will allow better performance.

Choose your XML document granularity with respect to the anticipated predominant granularity of access. When in doubt, it's usually better to lean toward finer granularity and smaller XML documents.

Use Fully Specified Paths in XPath Expressions

Assume that you have a table called customer that was created with one XML column called info. This command may have created the table:

```
create table customer(info XML);
```

Also assume that the XML to be saved in this table has the following format:

```
<customerinfo Cid="1004">
 <name>Matt Foreman</name>
 <addr country="Canada">
 <street>1596 Baseline</street>
 <city>Toronto</city>
 <state>Ontario</state>
 <pcode>M3Z-5H9</pcode>
 </addr>
 <phone type="work">905-555-4789</phone>
 <phone type="home">416-555-3376</phone>
</customerinfo>
```

If you want to retrieve customers' phone numbers, you can use either of the following XPath statements:

- `/customerinfo/phone`
- `//phone`

Likewise, if you want to retrieve the city your customers live in, you can use either of the following XPath statements:

- `/customerinfo/addr/city`
- `/customerinfo/*/city`

For best performance, the fully specified path is preferred over using * or // because it enables DB2 to navigate directly to the desired elements, skipping over nonrelevant parts of the document. In a nutshell, avoid * and // in your path expressions and use fully specified paths where possible.

Define Lean XML Indexes

Assume our queries often search for `customerinfo` documents by customer name. An index on the customer name element can greatly improve the performance of such queries.

Take a look at some sample code:

```
create table customer(info XML);
create index custname1 on customer(info)
generate key using xmlpattern
'/customerinfo/name' as sql varchar(20);
create index custname2 on customer(info)
generate key using xmlpattern '//name' as sql
varchar(20);
select * from customer
where xmlexists('$i/customerinfo[name = "Matt
Foreman"]' passing info as $i);
```

Both indexes defined in the preceding code are eligible to evaluate the `XMLEXISTS` predicate on the customer name. But index `custname2` can be substantially larger than index `custname1` because it contains index entries not only for customer names but also for assistant names. This size difference is because the XML pattern `//name` matches name elements anywhere in the document. However, if you never search by assistant name, then you don't need them indexed.

For read operations, index `custname1` is smaller and therefore potentially better performing. For `insert`, `update`, and `delete` operations, index `custname1` incurs index maintenance overhead only for customer names, while index `custname2` requires index maintenance for customer and assistant names. You certainly don't want to pay that extra price if you require maximum performance for `insert`, `update`, and `delete` operations and don't need indexed access based on assistant names.

Also, consider the following `heavyIndex`, which indexes everything:

```
create index heavyIndex on customer(info)
generate key using xmlpattern '//text()' as
sql varchar(20);
```

The preceding example indexes entries for every text node. In other words, it contains an index entry for every leaf element value in every XML document in the XML column.

We don't recommend this type of index because it's very costly to maintain during `insert`, `update`, and `delete` operations and consumes a lot of storage space. The only exception may be applications with low write activity and unpredictable query workload such that more specific indexes are hard to define.

In a nutshell, be as precise as possible when defining XML indexes and avoid `*` and `//` if you can.

Use RUNSTATS to Collect Statistics

The `RUNSTATS` utility has been extended to collect statistics on XML data and XML indexes. DB2's cost-based optimizer uses these statistics to generate efficient execution plans for XQuery and SQL/XML queries. Thus, continue to use `RUNSTATS` as you would for relational data.

If your table contains relational and XML data and you want to refresh the relational statistics only, you can execute `RUNSTATS` with the new clause `EXCLUDING XML COLUMNS`. Without this clause, the default and preferred behavior is to always collect statistics for relational and XML data.

For relational data as well as XML data, you can enable sampling to reduce the time for executing `RUNSTATS`. On a large data set, the statistics from 10 percent (or even less) of the data are often still very representative of the total population.

Whatever sampling percentage you choose, `RUNSTATS` allows you to sample rows (Bernoulli sampling) or pages (system sampling):

- Row-level sampling reads all data pages but considers only a percentage of the rows on each page. It's therefore only a subset of the corresponding XDA pages.

- Page level sampling reduces I/O significantly becase it reads only a percentage of the data pages. As a result, page sampling can significantly improve performance if your table contains not just XML but also a fair amount of relational data.

Row-level sampling may produce more accurate statistics if relational data values are highly clustered.

DB2 does generate better execution plans if XML statistics are available. Use `RUNSTATS` as you normally would or use `RUNSTATS` with sampling to reduce its execution time.

Use SQL/XML with Parameter Markers for Short Queries and OLTP Applications

Very short database queries often execute so fast that the time to compile and optimize them is a substantial portion of their total response time. Usually, you'll want to *compile* (prepare) these queries just once and only pass predicate literal values for each execution.

While DB2 XQuery doesn't support external parameters, the `SQL/XML` functions `XMLQUERY`, `XMLTABLE` and `XMLEXISTS` do. They allow you to pass SQL parameter markers as a variable into the embedded XQuery expressions. This method is recommended for applications with short and repetitive queries.

Short queries and `OLTP` transactions are faster as prepared statements with parameter markers. For XML, this optimization requires SQL/XML to pass SQL-style parameters to XQuery expressions.

Chapter 7

Ten DB2 pureXML Resources You Won't Want to Miss

In This Chapter

- Surfing the Web for free books
- Talking to others about DB2 pureXML

As you continue to work with DB2 pureXML, you may want to take advantage of the many online resources — even books — available to you. And the best news? The ten or so resources in this chapter are all free.

Get a Free Copy of DB2

Go to the following URL to download a free copy of DB2:

```
www.ibm.com/software/data/db2/express
```

The installation is a snap.

Read a Free Book or Two

To help you get started using DB2 with pureXML, IBM wrote an excellent free book called *DB2 9: pureXML Overview and Fast Start*.

```
www.redbooks.ibm.com/abstracts/sg247298.html
```

If you want to read even more about pureXML, then check out another free book, *DB2 9 pureXML Guide*. This book has almost 400 fun-filled pages of pureXML information.

www.redbooks.ibm.com/abstracts/sg247315.html

Discover the Latest DB2 News

The DB2 pureXML product Web page includes all the latest news about the product.

www.ibm.com/software/data/db2/xml/

Try a Few Tutorials

If you want to try your hand at DB2 pureXML, wiki is the place to go. It also has the latest articles about using pureXML.

www.ibm.com/developerworks/wikis/display/db2xml/

Get Answers to Your Questions

If you have questions about using DB2 pureXML and need a fast answer, visit the DB2 pureXML forum. You can always find helpful people who are more than happy to answer your questions.

www.ibm.com/developerworks/forums/
forum.jspa?forumID=1423&start=0

Network with Fellow pureXML Users

If you want to talk to other DB2 users, visit the pureXML user community over on Channel DB2.

www.channeldb2.ning.com/group/purexml

Blog a Little

You can get all the latest news directly from the keyboards of the team that develop DB2 pureXML at the pureXML Team Blog.

`www.ibm.com/developerworks/blogs/page/purexml`

Get a Little Personal — at My Blog, That Is

Last, but not least, my personal blog, called "Native XML Databases," covers various topics of interest to people working with native XML databases.

`www.nativexmldatabase.com`

Get Professional

If you're a member of LinkedIn, make sure to join the DB2 pureXML group:

`www.linkedin.com/groups?gid=129185`

Notes

Notes

pureXML enables true end-to-end XML data processing environments

Use DB2 pureXML to keep data in its "pure" XML format

When it comes to XML data, DB2 isn't like other relational databases. DB2 pureXML stores the XML data in its purest format. This is great for you, because it makes it very easy to work with the XML data. You don't need to transform the XML data into another format, and you don't need to parse the XML data after retrieval. You simply work directly with the XML data in the database.

Discover how to:

- *Store XML data*
- *Query XML data*
- *Improve performance*
- *Improve productivity*

THE DUMMIES WAY®

- *Explanations in plain English*
- *"Get in, get out" information*
- *Icons and other navigational aids*
- *Top ten lists*
- *A dash of humor and fun*

Get smart! @ www.dummies.com

- *Find listings of all our books*
- *Choose from many different subject categories*
- *Sign up for eTips at etips.dummies.com*

ISBN: 978-0-470-44057-5
Not resaleable

For Dummies®
A Branded Imprint of
WILEY