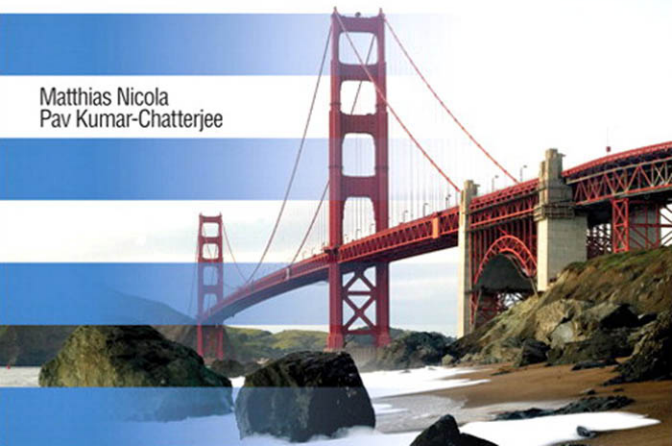


DB2 pureXML Cookbook

Master the Power of the
IBM Hybrid Data Server

Matthias Nicola
Pav Kumar-Chatterjee



The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein. Before you use any IBM or non-IBM or open-source product mentioned in this book, make sure that you accept and adhere to the licenses and terms and conditions for any such product.

© Copyright 2010 by International Business Machines Corporation. All rights reserved.

Note to U.S. Government Users: Documentation related to restricted right. Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.

IBM Press Program Managers: Steven M. Stansel, Ellice Uffer

Cover design: IBM Corporation

Associate Publisher: Greg Wiegand

Marketing Manager: Kournaye Sturgeon

Publicist: Heather Fox

Acquisitions Editor: Bernard Goodwin

Managing Editor: Kristy Hart

Designer: Alan Clements

Project Editor: Andy Beaster

Copy Editor: Paula Lowell

Senior Indexer: Cheryl Lenser

Compositor: Gloria Schurick

Proofreader: Leslie Joseph

Manufacturing Buyer: Dan Uhrig

Published by Pearson plc

Publishing as IBM Press

IBM Press offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales

1-800-382-3419

corpsales@pearsontechgroup.com.

For sales outside the U.S., please contact:

International Sales

international@pearson.com.

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both: IBM, the IBM logo, IBM Press, DB2, pureXML, z/OS, ibm.com, WebSphere, System z, developerWorks, InfoSphere, DRDA, Rational, AIX, OmniFind, i5/OS, Lotus, and DataPower. Microsoft, Windows, Microsoft Word, Microsoft Visual Studio, Visual Basic, and Visual C# 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. Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc., in the United States, other countries, or both. Other company, product, or service names may be trademarks or service marks of others.

Library of Congress Cataloging-in-Publication Data

Nicola, Matthias.

DB2 PureXML cookbook : master the power of IBM's hybrid data server / Matthias Nicola and Pav Kumar-Chatterjee.

p. cm.

Includes indexes.

ISBN-13: 978-0-13-815047-1 (hardback : alk. paper)

ISBN-10: 0-13-815047-8 (hardback : alk. paper) 1. IBM Database 2. XML (Document markup language) 3. Database management. I. Kumar-Chatterjee, Pav. II. Title.

QA76.9.D3N525 2009

006.7'4—dc22

2009020222

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671 3447

ISBN-13: 978-0-13-815047-1

ISBN-10: 0-13-815047-8

Text printed in the United States on recycled paper at Edwards Brothers in Ann Arbor, Michigan.
First printing August 2009

Foreword

In the years since E.F. Codd's groundbreaking work in the 1970s, relational database systems have become ubiquitous in the business world. Today, most of the world's business data is stored in the rows and columns of relational databases. The relational model is ideally suited to applications in which data has a relatively simple and uniform structure, and in which database structure evolves much more slowly than data values.

With the advent of the Web, however, big changes began to occur in the database world, driven by globalization and by dramatic reductions in the cost of storing, transmitting, and processing data. Today, businesses are globally interconnected and exchange large volumes of data with customers, suppliers, and governments. Much of this data consists of things that do not fit neatly into rows and columns, such as medical records, legal documents, incident reports, tax returns, and purchase orders. The new kinds of data tend to be more heterogeneous than traditional business data, having more variation and a more rapidly evolving structure.

In response to the changing requirements of business data, a new generation of standards have appeared. XML has emerged as an international standard for the exchange of self-describing data, unifying structured, unstructured, and semi-structured information formats. XML Schema has been adopted as the metadata syntax for describing the structure of XML documents. Industry-specific XML schemas have been developed for medical, insurance, retail, publishing, banking, and other industries. XPath and XQuery have been adopted as standard languages for retrieving and manipulating data in XML format, and new facilities have been added to the SQL standard for interfacing between relational and XML data.

In DB2, the new generation of XML-related standards is reflected in *pureXML*, a broad new set of XML functionality implemented in both DB2 for z/OS and DB2 for Linux, UNIX, and Windows. *pureXML* bridges the gap between the XML and relational worlds and makes DB2 a true hybrid database management system. DB2 *pureXML* stores and indexes XML data alongside relational data in a highly efficient new storage format, and supports XML query languages such as XPath and XQuery alongside the traditional SQL.

pureXML is perhaps the largest new package of functionality in the history of DB2, impacting nearly every aspect of the system. The implementation of *pureXML* required deep changes in the database kernel, optimization methods, database administrator tools, system utilities, and application programming interfaces. New facilities were added for registering XML schemas and using them to validate stored documents. New kinds of statistics on XML documents had to be gathered and exploited. Facilities for replicated, federated, and partitioned databases had to be updated to accommodate the new XML storage format.

pureXML provides DB2 users with a new level of capability, but using this capability to full advantage requires users to have a new level of sophistication. A new user of *pureXML* is

confronted with many complex choices. What kinds of data should be represented in XML rather than in normalized tables? How can data be converted between XML and relational formats? How can a hybrid database be designed to take advantage of both data formats? What are the most appropriate uses for SQL, XQuery, and XPath? What kinds of indexes should be maintained on XML data? What is the XML equivalent of a NULL value? These and many other questions are considered in detail in the *DB2 pureXML Cookbook*.

Matthias Nicola has been deeply involved in the design and implementation of DB2 pureXML since its inception. As a Senior Engineer at IBM's Silicon Valley Laboratory, his work has focused on measuring and optimizing the performance of new storage and indexing techniques for XML. After the release of pureXML, he worked with many IBM customers and business partners to create, deploy, and optimize XML applications for government, banking, telecommunications, retail, and other industries.

Pav Kumar-Chatterjee is a technical specialist with many years of experience in consulting with IBM customers throughout the UK and Europe on developing and deploying DB2 and XML solutions.

Through their work with customers, Matthias and Pav have learned how to explain concepts clearly and how to identify and avoid common pitfalls in the application development process. They have also developed a set of "best practices" that they have shared at numerous conferences, classes, workshops, and customer engagements. Between them, Matthias and Pav have accumulated all the knowledge and experience you need to successfully create and deploy solutions using DB2 pureXML. Their expertise is encapsulated in this book in the form of hundreds of practical examples, tested and clearly explained. The book also includes a comprehensive set of questions to test your understanding.

DB2 pureXML Cookbook includes both an introduction to basic XML concepts and a comprehensive description of the XML-related features of DB2 for z/OS and DB2 for Linux, UNIX, and Windows. Chapters are organized around tasks that reflect the lifecycle of XML projects, including designing databases, loading and validating data, writing queries and updates, developing applications, optimizing performance, and diagnosing problems. Each topic provides a clear progression from introductory material to more advanced concepts. The writing style is informal and easy to understand for both beginners and experts.

If you are an application developer, database administrator, or system architect, this is the book you need to gain a comprehensive understanding of DB2 pureXML.

Don Chamberlin
IBM Fellow, Emeritus
Almaden Research Center
April 10, 2009

Preface

In recent years XML has continued to emerge as the de-facto standard for data exchange, because it is flexible, extensible, self-describing, and suitable for any combination of structured and unstructured data. With the increasing use of XML as a pervasive data format, there is a growing need to store, index, query, update, and validate XML documents in database systems. In response to this demand, IBM has developed sophisticated XML data management capabilities that are deeply integrated in the DB2 database system. This novel technology is called *DB2 pureXML* and is available in DB2 for z/OS and DB2 for Linux, UNIX, and Windows. With pureXML, DB2 has evolved into a *hybrid* database system that allows you to manage both XML and relational data in a tightly integrated manner.

The *DB2 pureXML Cookbook* provides the single most comprehensive coverage of DB2's pureXML functionality in DB2 for Linux, UNIX, and Windows as well as DB2 for z/OS. This book is a “cookbook” because it is more than just a description of functions and features (“ingredients”). This book provides “recipes” that show you how to combine the pureXML ingredients to efficiently perform typical user tasks for managing XML data. This book explains DB2 pureXML in more than 700 practical examples, including 250+ XQuery and SQL/XML queries, taking you from simple introductions all the way to advanced scenarios, tuning, and troubleshooting.

Since the first release of DB2 pureXML in 2006 we have worked with numerous companies to help them design, implement, optimize, and deploy XML applications with DB2. In this book we have distilled our experience from these pureXML projects so that you can benefit from proven implementation techniques, best practices, tips and tricks, and performance guidelines that are not described elsewhere.

WHO SHOULD READ THIS BOOK?

This book is written for database administrators, application developers, IT architects, and everyone who wants to get a deep technical understanding of DB2's pureXML technology and how to use it most effectively. As a DBA you will learn, for example, how to design and manage XML storage objects, how to index XML data, where to find XML-related information in the DB2 catalog, and how to manage XML with DB2 utilities. Application developers learn, among other things, how to write XML queries and XML updates with XPath, SQL/XML, and XQuery, and how to code XML applications with Java, .NET, C, COBOL, PL/1, PHP, or Perl.

This book is suitable for both beginners and experts. Each topic starts with simple examples, which provide an easy introduction, and works towards advanced concepts and solutions to complex problems. Extensive XML knowledge is not required to read this book because it includes the necessary introductions to XML, XPath, XQuery, XML Schema, and namespaces. These

concepts are explained through numerous examples that are easy to follow. We assume that you have some experience with relational databases and SQL, but we show all the relevant DB2 commands that are required to work through the examples in this book. Appendix C, *Further Reading*, also contains links to additional educational material about both DB2 and XML.

COVERAGE OF DB2 FOR z/OS AND DB2 FOR LINUX, UNIX, AND WINDOWS IN THIS BOOK

The book describes DB2 pureXML on all supported platforms and versions, which at the time of writing are DB2 9 for z/OS as well as DB2 9.1, 9.5, and 9.7 for Linux, UNIX, and Windows. Many pureXML features and functions are identical across DB2 for Linux, UNIX, and Windows and DB2 for z/OS.

Where platform-specific differences exist we point them out along the way. However, this book does not intend to be a reference that lists all functions and features according to platform and version of DB2. Instead, this book is a “cookbook” that focuses on concepts, examples, and best practices. The capabilities in DB2 for z/OS and DB2 for Linux, UNIX, and Windows continue to grow and converge over time. For the latest information on which feature is available in which version, please consult the respective DB2 information center. DB2 for z/OS also continues to deliver pureXML enhancements via APARs. Please look at APAR II14426, which is an informational APAR that summarizes and links all other XML-related APARs for DB2 on z/OS.

In our work with users who adopt DB2 pureXML we have made the following observation: Some of the users who begin to use DB2 pureXML on Linux, UNIX, and Windows have little or no prior experience with DB2. In contrast, most users who are interested in DB2 pureXML on z/OS are already familiar with DB2 for z/OS in general. This difference is reflected in this book; that is, we describe some DB2 concepts, such as monitoring or the use of DB2 utilities, in more detail for DB2 for Linux, UNIX, and Windows than for DB2 for z/OS.

DO IT YOURSELF!

The best way to learn a new technology is hands-on. We strongly recommend that you download DB2 Express-C, which is free, and try the concepts that you learn in this book in DB2’s sample database. Appendixes A and B contain the necessary information to get you started.

DON’T HESITATE TO ASK QUESTIONS!

If any pureXML question is not covered in this book, the fastest way to get an answer is to post a question in the DB2 pureXML forum at <http://www.ibm.com/developerworks/forums/forum.jspa?forumID=1423>.

Whether you seek clarification about specific features or functions, or if you need help with a tricky query, this forum is the right place to ask for help. You are also welcome to contact the

authors directly. If you want to discuss an XML project or if you have comments or feedback on the material in this book—we will be happy to hear from you. Please contact Matthias at mnicola@us.ibm.com and Pav at kumarp2@uk.ibm.com.

HOW THIS BOOK IS STRUCTURED

The *DB2 pureXML Cookbook* takes you through the different tasks and topics that you typically encounter during the life cycle of an XML project. The structure of this book with its 23 chapters is the following:

Planning

Chapter 1, *Introduction*, provides an overview of XML and its differences to relational data, and discusses scenarios where XML has advantages over the relational model. This chapter also includes a summary of the pureXML technology.

Chapter 2, *Designing XML Data and Applications*, covers fundamental XML design questions such as choosing between XML elements and attributes, selecting an appropriate XML document granularity, and deciding on a “good” mix of XML and relational data for your application.

Designing and Populating an XML Database

Chapter 3, *Designing and Managing XML Storage Objects*, first explains the tree representation of XML documents and how they are physically stored in DB2. Then it describes how to create and manage tables and table spaces for XML, including compression, reorganization, and partitioning.

Chapter 4, *Inserting and Retrieving XML Data*, looks at “full document” operations such as insert, delete, and retrieval of XML documents. This chapter also explains how to handle XML declarations, white space, and reserved characters in XML documents.

Chapter 5, *Moving XML Data*, looks at importing, exporting, loading, replicating, and federating XML data in DB2. A technique to split large XML documents into smaller ones is also demonstrated.

Querying XML Data

Chapter 6, *Querying XML Data: Introduction and XPath*, is the first of four chapters on querying XML data. This chapter provides an overview of the different options for querying XML, introduces the XPath and XQuery data model, and describes the XPath language in detail. These concepts are fundamental for the subsequent chapters.

Chapter 7, *Querying XML Data with SQL/XML*, explains how XPath can be included in SQL statements with the SQL/XML functions `XMLQUERY` and `XMLTABLE` and the `XMLEXISTS` predicate. The use of SQL/XML is illustrated through a rich collection of examples and a discussion of common mistakes and how to avoid them.

Chapter 8, *Querying XML Data with XQuery*, introduces the XQuery language, which is a superset of XPath. Among other things, this chapter describes XQuery FLWOR expressions, combinations of SQL and XQuery, and a comparison of XPath, XQuery, and SQL/XML.

Chapter 9, *Querying XML Data: Advanced XML Queries and Troubleshooting*, takes querying XML data to the expert level. It demonstrates how to perform grouping, aggregation, and joins over XML data or a mix of XML and relational data. The troubleshooting section discusses “bad” XML queries, common errors, and how to avoid both.

Converting, Updating, and Transforming

Chapter 10, *Producing XML from Relational Data*, begins the discussion of converting, updating, and transforming data. This chapter explains how to read relational data from existing database tables and construct XML documents from it.

Chapter 11, *Converting XML to Relational Data*, describes the opposite of Chapter 10, that is, the process of decomposing or shredding XML documents into relational tables. Two shredding methods are discussed, one using the `XMLTABLE` function and the other using annotated XML Schemas.

Chapter 12, *Updating and Transforming XML Documents*, covers three techniques for updating XML documents: Full document replacement, XSLT transformations, and the XQuery Update Facility that allows you to modify, insert, delete, or rename individual elements and attributes within an XML document.

Performance and Monitoring

Chapter 13, *Defining and Using XML Indexes*, is one of two chapters dedicated to performance. It describes how to create XML indexes to improve query performance and explains under which conditions query predicates can or cannot use XML indexes.

Chapter 14, *Performance and Monitoring*, looks at analyzing the performance of XML operations with particular emphasis on understanding XML query access plans. A summary of best practices for XML performance in DB2 is also provided.

Ensuring Data Quality

Chapter 15, *Managing XML Data with Namespaces*, introduces XML namespaces and explains how they avoid naming conflicts and ambiguity, thus contributing to data quality. This chapter illustrates how to index, query, update, and construct XML documents that contain namespaces.

Chapter 16, *Managing XML Schemas*, first describes how XML Schemas can constrain XML documents in terms of their structure, element and attribute names, data types, and other characteristics. Then this chapter walks you through the concepts of registering, managing, and evolving XML Schemas in DB2.

Chapter 17, *Validating XML Documents against XML Schemas*, concentrates on the validation of XML documents to ensure XML data quality in DB2. You can validate XML documents in INSERT and UPDATE statements, queries, and import and load operations.

Application Development

Chapter 18, *Using XML in Stored Procedures, UDFs, and Triggers*, demonstrates how you can implement application-specific processing logic with XML manipulation in SQL stored procedures, user-defined functions, and triggers.

Chapter 19, *Performing Full-Text Search*, describes how the *DB2 Net Search Extender* and *DB2 Text Search* support efficient full-text search in collections of XML documents.

Chapter 20, *Understanding XML Data Encoding*, explains internal and external XML encoding, how DB2 determines and handles XML encoding, and how you can avoid code page conversion.

Chapter 21, *Developing XML Application with DB2*, contains techniques and best practices for application programs that exchange XML data with the DB2 server. Code samples are provided for Java, .NET, C, COBOL, PL/1, PHP, and Perl programmers.

Reference Material

Chapter 22, *Exploring XML Information in the DB2 Catalog*, is a guide to how XML storage objects, XML indexes, and XML Schemas are listed in the database catalog.

Chapter 23, *Test Your Knowledge—The DB2 pureXML Quiz*, offers 82 questions to revisit specific topic areas.

The Appendixes list supporting information and further reading for each chapter.

Converting XML to Relational Data

This chapter describes methods to convert XML documents to rows in relational tables. This conversion is commonly known as *shredding* or *decomposing* of XML documents. Given the rich support for XML columns in DB2 you might wonder in which cases it can still be useful or necessary to convert XML data to relational format. One common reason for shredding is that existing SQL applications might still require access to the data in relational format. For example, legacy applications, packaged business applications, or reporting software do not always understand XML and have fixed relational interfaces. Therefore you might sometimes find it useful to shred all or some of the data values of an incoming XML document into rows and columns of relational tables.

In this chapter you learn:

- The advantages and disadvantages of shredding and of different shredding methods (section 11.1)
- How to shred XML data to relational tables using `INSERT` statements that contain the `XMLTABLE` function (section 11.2)
- How to use XML Schema annotations that map and shred XML documents to relational tables (section 11.3)

11.1 ADVANTAGES AND DISADVANTAGES OF SHREDDING

The concept of XML shredding is illustrated in Figure 11.1. In this example, XML documents with customer name, address, and phone information are mapped to two relational tables. The documents can contain multiple phone elements because there is a one-to-many relationship

between customers and phones. Hence, phone numbers are shredded into a separate table. Each repeating element, such as phone, leads to an additional table in the relational target schema. Suppose the customer information can also contain multiple email addresses, multiple accounts, a list of most recent orders, multiple products per order, and other repeating items. The number of tables required in the relational target schema can increase very quickly. Shredding XML into a large number of tables can lead to a complex and unnatural fragmentation of your logical business objects that makes application development difficult and error-prone. Querying the shredded data or reassembling the original documents may require complex multiway joins.

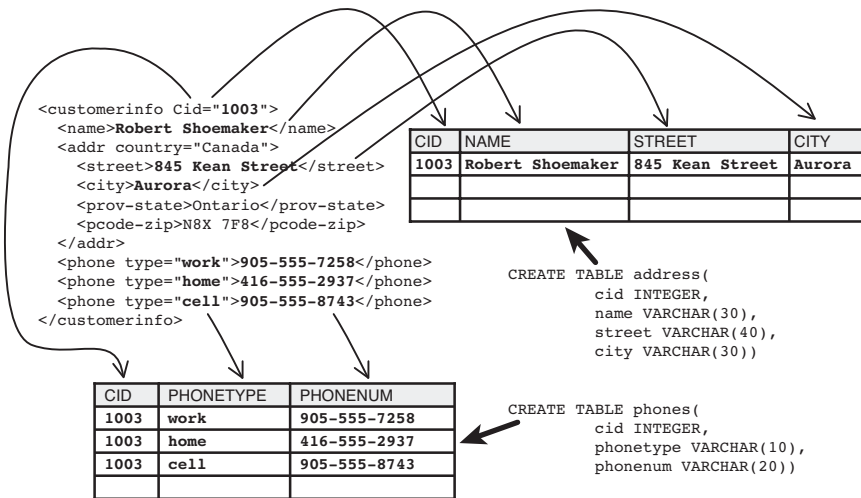


Figure 11.1 Shredding of an XML document

Depending on the complexity, variability, and purpose of your XML documents, shredding may or may not be a good option. Table 11.1 summarizes the pros and cons of shredding XML data to relational tables.

Table 11.1 When Shredding Is and Isn't a Good Option

Shredding Can Be Useful When...	Shredding Is Not A Good Option When...
<ul style="list-style-type: none"> • Incoming XML data is just feeding an existing relational database. • The XML documents do not represent logical business objects that should be preserved. • Your primary goal is to enable existing relational applications to access XML data. • You are happy with your relational schema and would like to use it as much as possible. 	<ul style="list-style-type: none"> • Your XML data is complex and nested, and difficult to map to a relational schema. • Mapping your XML format to a relational schema leads to a large number of tables. • Your XML Schema is highly variable or tends to change over time. • Your primary goal is to manage XML documents as intact business objects.

Table 11.1 When Shredding Is and Isn't a Good Option (*Continued*)

Shredding Can Be Useful When...	Shredding Is Not A Good Option When...
<ul style="list-style-type: none"> • The structure of your XML data is such that it can easily be mapped to relational tables. • Your XML format is relatively stable and changes to it are rare. • You rarely need to reconstruct the shredded documents. • Querying or updating the data with SQL is more important than insert performance. 	<ul style="list-style-type: none"> • You frequently need to reconstruct the shredded documents or parts of them. • Ingesting XML data into the database at a high rate is important for your application.

In many XML application scenarios the structure and usage of the XML data does not lend itself to easy and efficient shredding. This is the reason why DB2 supports XML columns that allow you to index and query XML data without conversion. Sometimes you will find that your application requirements can be best met with *partial shredding* or *hybrid XML storage*.

- *Partial shredding* means that only a subset of the elements or attributes from each incoming XML document are shredded into relational tables. This is useful if a relational application does not require *all* data values from each XML document. In cases where shredding each document entirely is difficult and requires a complex relational target schema, partial shredding can simplify the mapping to the relational schema significantly.
- *Hybrid XML storage* means that upon insert of an XML document into an XML column, selected element or attribute values are extracted and redundantly stored in relational columns.

If you choose to shred XML documents, entirely or partially, DB2 provides you with a rich set of capabilities to do some or all of the following:

- Perform custom transformations of the data values before insertion into relational columns.
- Shred the same element or attribute value into multiple columns of the same table or different tables.
- Shred multiple different elements or attributes into the same column of a table.
- Specify conditions that govern when certain elements are or are not shredded. For example, shred the address of a customer document only if the `country` is Canada.
- Validate XML documents with an XML Schema during shredding.
- Store the full XML document along with the shredded data.

DB2 9 for z/OS and DB2 9.x for Linux, UNIX, and Windows support two shredding methods:

- SQL `INSERT` statements that use the `XMLTABLE` function. This function navigates into an input document and produces one or multiple relational rows for insert into a relational table.
- Decomposition with an annotated XML Schema. Since an XML Schema defines the structure of XML documents, annotations can be added to the schema to define how elements and attributes are mapped to relational tables.

Table 11.2 and Table 11.3 discuss the advantages and disadvantages of the `XMLTABLE` method and the annotated schema method.

Table 11.2 Considerations for the `XMLTABLE` Method

Advantages of the <code>XMLTABLE</code> Method	Disadvantages of the <code>XMLTABLE</code> Method
<ul style="list-style-type: none"> • It allows you to shred data even if you do not have an XML Schema. • It does not require you to understand the XML Schema language or to understand schema annotations for decomposition. • It is generally easier to use than annotated schemas because it is based on SQL and XPath. • You can use familiar XPath, XQuery, or SQL functions and expressions to extract and optionally modify the data values. • It often requires no or little work during XML Schema evolution. • The shredding process can consume data from multiple XML and relational sources, if needed, such as values from DB2 sequences or look-up data from other relational tables. • It can often provide better performance than annotated schema decompositions. 	<ul style="list-style-type: none"> • For each target table that you want to shred into you need one <code>INSERT</code> statement. • You might have to combine multiple <code>INSERT</code> statements in a stored procedure. • There is no GUI support for implementing the <code>INSERT</code> statements and the required <code>XMLTABLE</code> functions. You need to be familiar with XPath and SQL/XML.

Table 11.3 Considerations for Annotated Schema Decomposition

Advantages of the Annotated Schema Method	Disadvantages of the Annotated Schema Method
<ul style="list-style-type: none"> • The mapping from XML to relational tables can be defined using a GUI in IBM Data Studio Developer. • If you shred complex XML data into a large number of tables, the coding effort can be lower than with the XMLTABLE approach. • It offers a bulk mode with detailed diagnostics if some documents fail to shred. 	<ul style="list-style-type: none"> • It does not allow shredding without an XML Schema. • You might have to manually copy annotations when you start using a new version of your XML Schema. • Despite the GUI support, you need to be familiar with the XML Schema language for all but simple shredding scenarios. • Annotating an XML Schema can be complex, if the schema itself is complex.

11.2 SHREDDING WITH THE XMLTABLE FUNCTION

The XMLTABLE function is an SQL table function that uses XQuery expressions to create relational rows from an XML input document. For details on the XMLTABLE function, see Chapter 7, *Querying XML Data with SQL/XML*. In this section we describe how to use the XMLTABLE function in an SQL INSERT statement to perform shredding. We use the shredding scenario in Figure 11.1 as an example.

The first step is to create the relational target tables, if they don't already exist. For the scenario in Figure 11.1 the target tables are defined as follows:

```
CREATE TABLE address(cid INTEGER, name VARCHAR(30),
                    street VARCHAR(40), city VARCHAR(30))

CREATE TABLE phones(cid INTEGER, phonetype VARCHAR(10),
                   phonenum VARCHAR(20))
```

Based on the definition of the target tables you construct the INSERT statements that shred incoming XML documents. The INSERT statements have to be of the form INSERT INTO ... SELECT ... FROM ... XMLTABLE, as shown in Figure 11.2. Each XMLTABLE function contains a parameter marker (“?”) through which an application can pass the XML document that is to be shredded. SQL typing rules require the parameter marker to be cast to the appropriate data type. The SELECT clause selects columns produced by the XMLTABLE function for insert into the address and phones tables, respectively.

```

INSERT INTO address(cid, name, street, city)
  SELECT x.custid, x.custname, x.str, x.place
  FROM XMLTABLE('$i/customerinfo' PASSING CAST(? AS XML) AS "i"
    COLUMNS
      custid      INTEGER      PATH '@Cid',
      custname    VARCHAR(30)  PATH 'name',
      str         VARCHAR(40)  PATH 'addr/street',
      place      VARCHAR(30)  PATH 'addr/city' ) AS x ;

INSERT INTO phones(cid, phonetype, phonenumber)
  SELECT x.custid, x.pctype, x.number
  FROM XMLTABLE('$i/customerinfo/phone'
    PASSING CAST(? AS XML) AS "i"
    COLUMNS
      custid      INTEGER      PATH '../@Cid',
      number      VARCHAR(15)  PATH '.',
      pctype      VARCHAR(10)  PATH './@type') AS x ;

```

Figure 11.2 Inserting XML element and attribute values into relational columns

To populate the two target tables as illustrated in Figure 11.1, both `INSERT` statements have to be executed with the same XML document as input. One approach is that the application issues both `INSERT` statements in one transaction and binds the same XML document to the parameter markers for both statements. This approach works well but can be optimized, because the same XML document is sent from the client to the server and parsed at the DB2 server twice, once for each `INSERT` statement. This overhead can be avoided by combining both `INSERT` statements in a single stored procedure. The application then only makes a single stored procedure call and passes the input document once, regardless of the number of `INSERT` statements in the stored procedure. Chapter 18, *Using XML in Stored Procedures, UDFs, and Triggers*, demonstrates such a stored procedure as well as other examples of manipulating XML data in stored procedures and user-defined functions.

Alternatively, the `INSERT` statements in Figure 11.2 can read a set of input documents from an XML column. Suppose the documents have been loaded into the XML column `info` of the `customer` table. Then you need to modify one line in each of the `INSERT` statements in Figure 11.2 to read the input document from the `customer` table:

```

FROM customer, XMLTABLE('$i/customerinfo' PASSING info AS "i"

```

Loading the input documents into a staging table can be advantageous if you have to shred many documents. The `LOAD` utility parallelizes the parsing of XML documents, which reduces the time to move the documents into the database. When the documents are stored in an XML column in parsed format, the `XMLTABLE` function can shred the documents *without* XML parsing.

The `INSERT` statements can be enriched with XQuery or SQL functions or joins to tailor the shredding process to specific requirements. Figure 11.3 provides an example. The `SELECT` clause

contains the function `RTRIM` to remove trailing blanks from the column `x.ptype`. The row-generating expression of the `XMLTABLE` function contains a predicate that excludes home phone numbers from being shredded into the target table. The column-generating expression for the phone numbers uses the XQuery function `normalize-space`, which strips leading and trailing whitespace and replaces each internal sequence of whitespace characters with a single blank character. The statement also performs a join to the lookup table `areacodes` so that a phone number is inserted into the `phones` table only if its area code is listed in the `areacodes` table.

```
INSERT INTO phones(cid, phonetype, phonenumber)
  SELECT x.custid, RTRIM(x.ptype), x.number
  FROM areacodes a,
       XMLTABLE('$i/customerinfo/phone[@type != "home"]'
                PASSING CAST(? AS XML) AS "i"
       COLUMNS
         custid    INTEGER    PATH './@Cid',
         number    VARCHAR(15) PATH 'normalize-space(.)',
         ptype     VARCHAR(10) PATH './@type') AS x
  WHERE SUBSTR(x.number,1,3) = a.code;
```

Figure 11.3 Using functions and joins to customize the shredding

11.2.1 Hybrid XML Storage

In many situations the complexity of the XML document structures makes shredding difficult, inefficient, and undesirable. Besides the performance penalty of shredding, scattering the values of an XML document across a large number of tables can make it difficult for an application developer to understand and query the data. To improve XML insert performance and to reduce the number of tables in your database, you may want to store XML documents in a *hybrid* manner. This approach extracts the values of selected XML elements or attributes and stores them in relational columns alongside the full XML document.

The example in the previous section used two tables, `address` and `phones`, as the target tables for shredding the customer documents. You might prefer to use just a single table that contains the customer `cid`, `name`, and `city` values in relational columns and the full XML document with the repeating phone elements and other information in an XML column. You can define the following table:

```
CREATE TABLE hybrid(cid INTEGER NOT NULL PRIMARY KEY,
                    name VARCHAR(30), city VARCHAR(25), info XML)
```

Figure 11.4 shows the `INSERT` statement to populate this table. The `XMLTABLE` function takes an XML document as input via a parameter marker. The column definitions in the `XMLTABLE` function produce four columns that match the definition of the target table `hybrid`. The row-generating expression in the `XMLTABLE` function is just `$i`, which produces the full input document. This expression is the input for the column-generating expressions in the `COLUMNS` clause of the `XMLTABLE` function. In particular, the column expression `'.'` returns the full input

document as-is and produces the XML column `doc` for insert into the `info` column of the target table.

```
INSERT INTO hybrid(cid, name, city, info)
SELECT x.custid, x.custname, x.city, x.doc
FROM XMLTABLE('$i' PASSING CAST(? AS XML) AS "i"
  COLUMNS
    custid INTEGER      PATH 'customerinfo/@Cid',
    custname VARCHAR(30) PATH 'customerinfo/name',
    city   VARCHAR(25)  PATH 'customerinfo/addr/city',
    doc    XML          PATH '.' ) AS x;
```

Figure 11.4 Storing an XML document in a hybrid fashion

It is currently not possible to define check constraints in DB2 to enforce the integrity between relational columns and values in an XML document in the same row. You can, however, define INSERT and UPDATE triggers on the table to populate the relational columns automatically whenever a document is inserted or updated. Triggers are discussed in Chapter 18, *Using XML in Stored Procedures, UDFs, and Triggers*.

It can be useful to test such INSERT statements in the DB2 Command Line Processor (CLP). For this purpose you can replace the parameter marker with a literal XML document as shown in Figure 11.5. The literal document is a string that must be enclosed in single quotes and converted to the data type XML with the XMLPARSE function. Alternatively, you can read the input document from the file system with one of the UDFs that were introduced in Chapter 4, *Inserting and Retrieving XML Data*. The use of a UDF is demonstrated in Figure 11.6.

```
INSERT INTO hybrid(cid, name, city, info)
SELECT x.custid, x.custname, x.city, x.doc
FROM XMLTABLE('$i' PASSING
  XMLPARSE(document
    '<customerinfo Cid="1001">
      <name>Kathy Smith</name>
      <addr country="Canada">
        <street>25 EastCreek</street>
        <city>Markham</city>
        <prov-state>Ontario</prov-state>
        <pcode-zip>N9C 3T6</pcode-zip>
      </addr>
      <phone type="work">905-555-7258</phone>
    </customerinfo>') AS "i"
  COLUMNS
    custid INTEGER      PATH 'customerinfo/@Cid',
    custname VARCHAR(30) PATH 'customerinfo/name',
    city   VARCHAR(25)  PATH 'customerinfo/addr/city',
    doc    XML          PATH '.' ) AS x;
```

Figure 11.5 Hybrid insert statement with a literal XML document

```

INSERT INTO hybrid(cid, name, city, info)
  SELECT x.custid, x.custname, x.city, x.doc
  FROM XMLTABLE('$i' PASSING
    XMLPARSE(document
      blobFromFile('/xml/mydata/cust0037.xml')) AS "i"
    COLUMNS
      custid    INTEGER    PATH 'customerinfo/@Cid',
      custname  VARCHAR(30) PATH 'customerinfo/name',
      city      VARCHAR(25) PATH 'customerinfo/addr/city',
      doc       XML        PATH '.' ) AS x;

```

Figure 11.6 Hybrid insert statement with a “FromFile” UDF

The insert logic in Figure 11.4, Figure 11.5, and Figure 11.6 is identical. The only difference is how the input document is provided: via a parameter marker, as a literal string that is enclosed in single quotes, or via a UDF that reads a document from the file system.

11.2.2 Relational Views over XML Data

You can create relational views over XML data using XMLTABLE expressions. This allows you to provide applications with a relational or hybrid view of the XML data without actually storing the data in a relational or hybrid format. This can be useful if you want to avoid the overhead of converting large amounts of XML data to relational format. The basic SELECT ... FROM ... XMLTABLE constructs that were used in the INSERT statements in the previous section can also be used in CREATE VIEW statements.

As an example, suppose you want to create a relational view over the elements of the XML documents in the customer table to expose the customer identifier, name, street, and city values. Figure 11.7 shows the corresponding view definition plus an SQL query against the view.

```

CREATE VIEW custview(id, name, street, city)
AS
  SELECT x.custid, x.custname, x.str, x.place
  FROM customer,
    XMLTABLE('$i/customerinfo' PASSING info AS "i"
      COLUMNS
        custid    INTEGER    PATH '@Cid',
        custname  VARCHAR(30) PATH 'name',
        str       VARCHAR(40) PATH 'addr/street',
        place     VARCHAR(30) PATH 'addr/city' ) AS x;

SELECT id, name FROM custview WHERE city = 'Aurora';

ID          NAME
-----
1003 Robert Shoemaker

1 record(s) selected.

```

Figure 11.7 Creating a view over XML data

The query over the view in Figure 11.7 contains an SQL predicate for the `city` column in the view. The values in the `city` column come from an XML element in the underlying XML column. You can speed up this query by creating an XML index on `/customerinfo/addr/city` for the `info` column of the `customer` table. DB2 9 for z/OS and DB2 9.7 for Linux, UNIX, and Windows are able to convert the relational predicate `city = 'Aurora'` into an XML predicate on the underlying XML column so that the XML index can be used. This is not possible in DB2 9.1 and DB2 9.5 for Linux, UNIX, and Windows. In these previous versions of DB2, include the XML column in the view definition and write the search condition as an XML predicate, as in the following query. Otherwise an XML index cannot be used.

```
SELECT id, name
FROM custview
WHERE XMLEXISTS('$INFO/customerinfo/addr[city = "Aurora"]')
```

11.3 SHREDDING WITH ANNOTATED XML SCHEMAS

This section describes another approach to shredding XML documents into relational tables. The approach is called *annotated schema shredding* or *annotated schema decomposition* because it is based on annotations in an XML Schema. These annotations define how XML elements and attributes in your XML data map to columns in your relational tables.

To perform annotated schema shredding, take the following steps:

- Identify or create the relational target tables that will hold the shredded data.
- Annotate your XML Schema to define the mapping from XML to the relational tables.
- Register the XML Schema in the DB2 XML Schema Repository.
- Shred XML documents with Command Line Processor commands or built-in stored procedures.

Assuming you have defined the relational tables that you want to shred into, let's look at annotating an XML Schema.

11.3.1 Annotating an XML Schema

Schema annotations are additional elements and attributes in an XML Schema to provide mapping information. DB2 can use this information to shred XML documents to relational tables. The annotations do not change the semantics of the original XML Schema. If a document is valid for the annotated schema then it is also valid for the original schema, and vice versa. You can use an annotated schema to validate XML documents just like the original XML Schema. For an introduction to XML Schemas, see Chapter 16, *Managing XML Schemas*.

The following is one line from an XML Schema:

```
<xs:element name="street" type="xs:string" minOccurs="1"/>
```

This line defines an XML element called `street` and declares that its data type is `xs:string` and that this element has to occur at least once. You can add a simple annotation to this element definition to indicate that the element should be shredded into the column `STREET` of the table `ADDRESS`. The annotation consists of two additional attributes in the element definition, as follows:

```
<xs:element name="street" type="xs:string" minOccurs="1"
  db2-xdb:rowSet="ADDRESS" db2-xdb:column="STREET" />
```

The same annotation can also be provided as schema elements instead of attributes, as shown next. You will see later in Figure 11.8 why this can be useful.

```
<xs:element name="street" type="xs:string" minOccurs="1">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>ADDRESS</db2-xdb:rowSet>
        <db2-xdb:column>STREET</db2-xdb:column>
      </db2-xdb:rowSetMapping>
    </xs:appinfo>
  </xs:annotation>
</xs:element/>
```

The prefix `xs` is used for all constructs that belong to the XML Schema language, and the prefix `db2-xdb` is used for all DB2-specific schema annotations. This provides a clear distinction and ensures that the annotated schema validates the same XML documents as the original schema.

There are 14 different types of annotations. They allow you to specify what to shred, where to shred to, how to filter or transform the shredded data, and in which order to execute inserts into the target tables. Table 11.4 provides an overview of the available annotations, broken down into logical groupings by user task. The individual annotations are further described in Table 11.5.

Table 11.4 Overview and Grouping of Schema Annotations

If You Want to	Use This Annotation
Specify the target tables to shred into	<code>db2-xdb:rowSet</code> <code>db2-xdb:column</code> <code>db2-xdb:SQLSchema</code> <code>db2-xdb:defaultSQLSchema</code>
Specify what to shred	<code>db2-xdb:contentHandling</code>
Transform data values while shredding	<code>db2-xdb:expression</code> <code>db2-xdb:normalization</code> <code>db2-xdb:truncate</code>
Filter data	<code>db2-xdb:condition</code> <code>db2-xdb:locationPath</code>

(continues)

Table 11.4 Overview and Grouping of Schema Annotations (*Continued*)

If You Want to	Use This Annotation
Map an element or attribute to multiple columns	<code>db2-xdb:rowSetMapping</code>
Map several elements or attributes to the same column	<code>db2-xdb:table</code>
Define the order in which rows are inserted into the target table, to avoid referential integrity violations	<code>db2-xdb:rowSetOperationOrder</code> <code>db2-xdb:order</code>

Table 11.5 XML Schema Annotations

Annotation	Description
<code>db2-xdb:defaultSQLSchema</code>	The default relational schema for the target tables.
<code>db2-xdb:SQLSchema</code>	Overrides the default schema for individual tables.
<code>db2-xdb:rowSet</code>	The table name that the element or attribute is mapped to
<code>db2-xdb:column</code>	The column name that the element or attribute is mapped to.
<code>db2-xdb:contentHandling</code>	For an XML element, this annotation defines how to derive the value that will be inserted into the target column. You can chose the text value of just this element (<code>text</code>), the concatenation of this element's text and the text of all its descendant nodes (<code>stringValue</code>), or the serialized XML (including all tages) of this element and all descendants (<code>serializeSubtree</code>). If you omit this annotation, DB2 chooses an appropriate default based on the nature of the respective element.
<code>db2-xdb:truncate</code>	Specifies whether a value should be truncated if its length is greater than the length of the target column.
<code>db2-xdb:normalization</code>	Specifies how to treat whitespace—valid values are <code>whitespaceStrip</code> , <code>canonical</code> , and <code>original</code>
<code>db2-xdb:expression</code>	Specifies an expression that is to be applied to the data before insertion into the target table.

Table 11.5 XML Schema Annotations (*Continued*)

Annotation	Description
<code>db2-xdb:locationPath</code>	Filters based on the XML context. For example, if it is a customer address then shred to the cust table; if it is an employee address then shred to the employee table.
<code>db2-xdb:condition</code>	Specifies value conditions so that data is inserted into a target table only if all conditions are true.
<code>db2-xdb:rowSetMapping</code>	Enables users to specify multiple mappings, to the same or different tables, for an element or attribute.
<code>db2-xdb:table</code>	Maps multiple elements or attributes to a single column.
<code>db2-xdb:order</code>	Specifies the insertion order of rows among multiple tables.
<code>db2-xdb:rowSetOperationOrder</code>	Groups together multiple <code>db2-xdb:order</code> annotations.

To demonstrate annotated schema decomposition we use the shredding scenario in Figure 11.1 as an example. Assume that the target tables have been defined as shown in Figure 11.1. An annotated schema that defines the desired mapping is provided in Figure 11.8. Let's look at the lines that are highlighted in bold font. The first bold line declares the namespace prefix `db2-xdb`, which is used throughout the schema to distinguish DB2-specific annotations from regular XML Schema tags. The first use of this prefix is in the annotation `db2-xdb:defaultSQLSchema`, which defines the relational schema of the target tables. The next annotation occurs in the definition of the element name. The two annotation attributes `db2-xdb:rowSet="ADDRESS"` and `db2-xdb:column="NAME"` define the target table and column for the name element. Similarly, the `street` and `city` elements are also mapped to respective columns of the `ADDRESS` table. The next two annotations map the phone number and the `type` attribute to columns in the `PHONES` table. The last block of annotations belongs to the XML Schema definition of the `Cid` attribute. Since the `Cid` attribute value becomes the join key between the `ADDRESS` and the `PHONE` table, it has to be mapped to both tables. Two row set mappings are necessary, which requires the use of *annotation elements* instead of *annotation attributes*. The first `db2-xdb:rowSetMapping` maps the `Cid` attribute to the `CID` column in the `ADDRESS` table. The second `db2-xdb:rowSetMapping` assigns the `Cid` attribute to the `CID` column in the `PHONES` table.

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1" >
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>db2admin</db2-xdb:defaultSQLSchema>
    </xs:appinfo>
  </xs:annotation>

```

Figure 11.8 Annotated schema to implement the shredding in Figure 11.1 (*continues*)

```

<xs:element name="customerinfo">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" minOccurs="1"
        db2-xdb:rowSet="ADDRESS" db2-xdb:column="NAME" />
      <xs:element name="addr" minOccurs="1"
        maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="street" type="xs:string"
              minOccurs="1" db2-xdb:rowSet="ADDRESS"
              db2-xdb:column="STREET" />
            <xs:element name="city" type="xs:string"
              minOccurs="1" db2-xdb:rowSet="ADDRESS"
              db2-xdb:column="CITY" />
            <xs:element name="prov-state" type="xs:string"
              minOccurs="1" />
            <xs:element name="pcode-zip" type="xs:string"
              minOccurs="1" />
          </xs:sequence>
          <xs:attribute name="country" type="xs:string" />
        </xs:complexType>
      </xs:element>
      <xs:element name="phone" minOccurs="0"
        maxOccurs="unbounded" db2-xdb:rowSet="PHONES"
        db2-xdb:column="PHONENUM">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute name="type" form="unqualified"
                type="xs:string" db2-xdb:rowSet="PHONES"
                db2-xdb:column="PHONETYPE" />
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="Cid" type="xs:integer">
      <xs:annotation>
        <xs:appinfo>
          <db2-xdb:rowSetMapping>
            <db2-xdb:rowSet>ADDRESS</db2-xdb:rowSet>
            <db2-xdb:column>CID</db2-xdb:column>
          </db2-xdb:rowSetMapping>
          <db2-xdb:rowSetMapping>
            <db2-xdb:rowSet>PHONES</db2-xdb:rowSet>
            <db2-xdb:column>CID</db2-xdb:column>
          </db2-xdb:rowSetMapping>
        </xs:appinfo>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>
</xs:schema>

```

Figure 11.8 Annotated schema to implement the shredding in Figure 11.1 (*Continued*)

11.3.2 Defining Schema Annotations Visually in IBM Data Studio

You can add annotations to an XML Schema manually, using any text editor or XML Schema editor. Alternatively, you can use the Annotated XSD Mapping Editor in IBM Data Studio Developer. To invoke the editor, right-click on an XML Schema name and select **Open With, Annotated XSD Mapping Editor**. A screenshot of the mapping editor is shown in Figure 11.9. The left side of the editor shows the hierarchical document structure defined by the XML Schema (**Source**). The right side shows the tables and columns of the relational target schema (**Target**). You can add mapping relationships by connecting source items with target columns. There is also a discover function to find probable relationships. Mapped relationships are represented in the mapping editor by lines drawn between source elements and target columns.

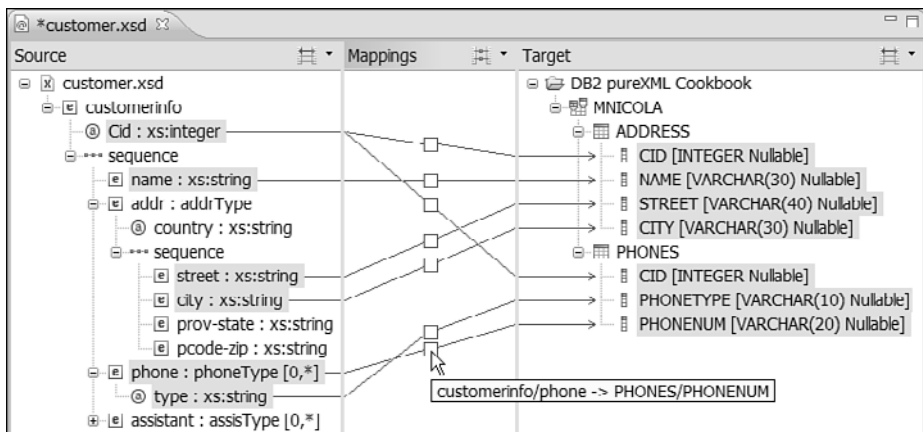


Figure 11.9 Annotated XSD Mapping Editor in Data Studio Developer

11.3.3 Registering an Annotated Schema

After you have created your annotated XML Schema you need to register it in the XML Schema Repository of the database. DB2's XML Schema Repository is described in detail in Chapter 16, *Managing XML Schemas*. For the annotated schema in Figure 11.8 it is sufficient to issue the `REGISTER XMLSCHEMA` command with its `COMPLETE` and `ENABLE DECOMPOSITION` options as shown in Figure 11.10. In this example the XML Schema is assumed to reside in the file `/xml/myschemas/cust2.xsd`. Upon registration it is assigned the SQL identifier `db2admin.cust2xsd`. This identifier can be used to reference the schema later. The `COMPLETE` option of the command indicates that there are no additional XML Schema documents to be added. The option `ENABLE DECOMPOSITION` indicates that this XML Schema can be used not only for document validation but also for shredding.

```
REGISTER XMLSCHEMA 'http://pureXMLcookbook.org'
FROM '/xml/myschemas/cust2.xsd'
AS db2admin.cust2xsd COMPLETE ENABLE DECOMPOSITION;
```

Figure 11.10 Registering an annotated XML schema

Figure 11.11 shows that you can query the DB2 catalog view `syscat.xsobjects` to determine whether a registered schema is enabled for decomposition (Y) or not (N).

```
SELECT SUBSTR(objectname,1,10) AS objectname,
       status, decomposition
FROM syscat.xsobjects ;
```

OBJECTNAME	STATUS	DECOMPOSITION
CUST2XSD	C	Y

Figure 11.11 Checking the status of an annotated XML schema

The `DECOMPOSITION` status of an annotated schema is automatically changed to `X` (*inoperative*) and shredding is disabled, if any of the target tables are dropped or a target column is altered. No warning is issued when that happens and subsequent attempts to use the schema for shredding fail. You can also use the following commands to disable and enable an annotated schema for shredding:

```
ALTER XSROBJECT cust2xsd DISABLE DECOMPOSITION;
ALTER XSROBJECT cust2xsd ENABLE DECOMPOSITION;
```

11.3.4 Decomposing One XML Document at a Time

After you have registered and enabled the annotated XML Schema you can decompose XML documents with the `DECOMPOSE XML DOCUMENT` command or with a built-in stored procedure. The `DECOMPOSE XML DOCUMENT` command is convenient to use in the DB2 Command Line Processor (CLP) while the stored procedure can be called from an application program or the CLP. The CLP command takes two parameters as input: the filename of the XML document that is to be shredded and the SQL identifier of the annotated schema, as in the following example:

```
DECOMPOSE XML DOCUMENT /xml/mydocuments/cust01.xml
XMLSCHEMA db2admin.cust2xsd VALIDATE;
```

The keyword `VALIDATE` is optional and indicates whether XML documents should be validated against the schema as part of the shredding process. While shredding, DB2 traverses both the XML document and the annotated schema and detects fundamental schema violations even if the `VALIDATE` keyword is not specified. For example, the shredding process fails with an error if a

mandatory element is missing, even if this element is not being shredded and the `VALIDATE` keyword is omitted. Similarly, extraneous elements or data type violations also cause the decomposition to fail. The reason is that the shredding process walks through the annotated XML Schema and the instance document in lockstep and therefore detects many schema violations “for free” even if the XML parser does not perform validation.

To decompose XML documents from an application program, use the stored procedure `XDBDECOMPXML`. The parameters of this stored procedure are shown in Figure 11.12 and described in Table 11.6.

```
>>-XDBDECOMPXML--(--rschema--,--xmlschemaname--,--xmldoc--,---->
>--documentid--,--validation--,--reserved--,--reserved--,----->
>--reserved--)-----><
```

Figure 11.12 Syntax and parameters of the stored procedure `XDBDECOMPXML`

Table 11.6 Description of the Parameters of the Stored Procedure `XDBDECOMPXML`

Parameter	Description
<code>rschema</code>	The relational schema part of the two-part SQL identifier of the annotated XML Schema. For example, if the SQL identifier of the XML Schema is <code>db2admin.cust2xsd</code> , then you should pass the string 'db2admin' to this parameter. In DB2 for z/OS this value must be either 'SYSXSR' or NULL.
<code>xmlschemaname</code>	The second part of the two-part SQL identifier of the annotated XML Schema. If the SQL identifier of the XML Schema is <code>db2admin.cust2xsd</code> , then you pass the string 'cust2xsd' to this parameter. This value cannot be NULL.
<code>xmldoc</code>	In DB2 for Linux, UNIX, and Windows, this parameter is of type <code>BLOB(1M)</code> and takes the XML document to be decomposed. In DB2 for z/OS this parameter is of type <code>CLOB AS LOCATOR</code> . This parameter cannot be NULL.
<code>documentid</code>	A string that the caller can use to identify the input XML document. The value provided will be substituted for any use of <code>\$DECOMP_DOCUMENTID</code> specified in the <code>db2-xdb:expression</code> or <code>db2-xdb:condition</code> annotations.
<code>validation</code>	Possible values are: 0 (no validation) and 1 (validation is performed). This parameter does not exist in DB2 for z/OS.
<code>reserved</code>	Parameters reserved for future use. The values passed for these arguments must be NULL. These parameters do not exist in DB2 for z/OS.

A Java code snippet that calls the stored procedure using parameter markers is shown in Figure 11.13

```
CallableStatement callStmt = con.prepareCall(
    "call SYSPROC.XDBDECOMPXML(?,?,?,?,?, null, null, null)");

File xmldoc = new File("c:\\mydoc.xml");
FileInputStream xmldocis = new FileInputStream(xmldoc);

callStmt.setString(1, "db2admin" );
callStmt.setString(2, "cust2xsd" );

// document to be shredded:
callStmt.setBinaryStream(3,xmldocis,(int)xmldoc.length() );

callStmt.setString(4, "mydocument26580" );

// no schema validation in this call:
callStmt.setInt(5, 0);

callStmt.execute();
```

Figure 11.13 Java code that invokes the stored procedure XDBDECOMPXML

While the input parameter for XML documents is of type CLOB AS LOCATOR in DB2 for z/OS, it is of type BLOB (1M) in DB2 for Linux, UNIX, and Windows. If you expect your XML documents to be larger than 1MB, use one of the stored procedures listed in Table 11.7. These stored procedures are all identical except for their name and the size of the input parameter `xmldoc`. When you call a stored procedure, DB2 allocates memory according to the *declared* size of the input parameters. For example, if all of your input documents are at most 10MB in size, the stored procedure XDBDECOMPXML10MB is a good choice to conserve memory.

Table 11.7 Stored Procedures for Different Document Sizes (DB2 for Linux, UNIX, and Windows)

Stored Procedure	Document Size	Supported since
XDBDECOMPXML	≤1MB	DB2 9.1
XDBDECOMPXML10MB	≤10MB	DB2 9.1
XDBDECOMPXML25MB	≤25MB	DB2 9.1
XDBDECOMPXML50MB	≤50MB	DB2 9.1
XDBDECOMPXML75MB	≤75MB	DB2 9.1
XDBDECOMPXML100MB	≤100MB	DB2 9.1
XDBDECOMPXML500MB	≤500MB	DB2 9.5 FP3

Table 11.7 Stored Procedures for Different Document Sizes (DB2 for Linux, UNIX, and Windows) (*Continued*)

Stored Procedure	Document Size	Supported since
XDBDECOMPXML1GB	≤1GB	DB2 9.5 FP3
XDBDECOMPXML1_5GB	≤1.5GB	DB2 9.7
XDBDECOMPXML2GB	≤2GB	DB2 9.7

For platform compatibility, DB2 for z/OS supports the procedure XDBDECOMPXML100MB with the same parameters as DB2 for Linux, UNIX, and Windows, including the parameter for validation.

11.3.5 Decomposing XML Documents in Bulk

DB2 9.7 for Linux, UNIX, and Windows introduces a new stored procedure called XDB_DECOMP_XML_FROM_QUERY. It uses an annotated schema to decompose one or multiple XML documents selected from a column of type XML, BLOB, or VARCHAR FOR BIT DATA. The main difference to the procedure XDBDECOMPXML is that XDB_DECOMP_XML_FROM_QUERY takes an SQL query as a parameter and executes it to obtain the input documents from a DB2 table. For a large number of documents, a LOAD operation followed by a “bulk decomp” can be more efficient than shredding these documents with a separate stored procedure call for each document. Figure 11.14 shows the parameters of this stored procedure. The parameters `commit_count` and `allow_access` are similar to the corresponding parameters of DB2’s `IMPORT` utility. The parameters `total_docs`, `num_docs_decomposed`, and `result_report` are output parameters that provide information about the outcome of the bulk shredding process. All parameters are explained in Table 11.8.

```
>>--XDB_DECOMP_XML_FROM_QUERY--(--rschema--/--xmlschema--/-->
>--query--/--validation--/--commit_count--/--allow_access--/-->
>--reserved--/--reserved2--/--continue_on_error--/-->
>--total_docs--/--num_docs_decomposed--/--result_report--)--<<
```

Figure 11.14 The stored procedure XDB_DECOMP_XML_FROM_QUERY

Table 11.8 Parameters for XDB_DECOMP_XML_FROM_QUERY

Parameter	Description
rschema	Same as for XDBDECOMPXML.
xmlschema	Same as xmlschemaname for XDBDECOMPXML.
query	A query string of type CLOB (1GB), which cannot be NULL. The query must be an SQL or SQL/XML SELECT statement and must return two columns. The first column must contain a unique document identifier for each XML document in the second column of the result set. The second column contains the XML documents to be shredded and must be of type XML, BLOB, VARCHAR FOR BIT DATA, or LONG VARCHAR FOR BIT DATA.
validation	Possible values are: 0 (no validation) and 1 (validation is performed).
commit_count	An integer value equal to or greater than 0. A value of 0 means the stored procedure does not perform any commits. A value of n means that a commit is performed after every n successful document decompositions.
allow_access	A value of 1 or 0. If the value is 0, then the stored procedure acquires an exclusive lock on all tables that are referenced in the annotated XML Schema. If the value is 1, then the stored procedure acquires a shared lock.
reserved, reserved2	These parameters are reserved for future use and must be NULL.
continue_on_error	Can be 1 or 0. A value of 0 means the procedure stops upon the first document that cannot be decomposed; for example, if the document does not match the XML Schema.
total_docs	An output parameter that indicates the total number of documents that the procedure <i>tried</i> to decompose.
num_docs_decomposed	An output parameter that indicates the number of documents that were <i>successfully</i> decomposed.
result_report	An output parameter of type BLOB (2GB). It contains an XML document that provides diagnostic information for each document that was not successfully decomposed. This report is not generated if all documents shredded successfully. The reason this is a BLOB field (rather than CLOB) is to avoid codepage conversion and potential truncation/data loss if the application code page is materially different from the database codepage.

Figure 11.15 shows an invocation of the XDB_DECOMP_XML_FROM_QUERY stored procedure in the CLP. This stored procedure call reads all XML documents from the `info` column of the `customer` table and shreds them with the annotated XML Schema `db2admin.cust2xsd`. The procedure commits every 25 documents and does not stop if a document cannot be shredded.

```

call SYSPROC.XDB_DECOMP_XML_FROM_QUERY
('DB2ADMIN', 'CUST2XSD', 'SELECT cid, info FROM customer',
 0, 25, 1, NULL, NULL, '1',?,?,?) ;

Value of output parameters
-----
Parameter Name : TOTALDOCS
Parameter Value : 100

Parameter Name : NUMDOCSDECOMPOSED
Parameter Value : 100

Parameter Name : RESULTREPORT
Parameter Value : x'

Return Status = 0

```

Figure 11.15 Calling the procedure `SYSPROC.XDB_DECOMP_XML_FROM_QUERY`

If you frequently perform bulk shredding in the CLP, use the command `DECOMPOSE XML DOCUMENTS` instead of the stored procedure. It is more convenient for command-line use and performs the same job as the stored procedure `XDB_DECOMP_XML_FROM_QUERY`. Figure 11.16 shows the syntax of the command. The various clauses and keywords of the command have the same meaning as the corresponding stored procedure parameters. For example, **query** is the `SELECT` statement that provides the input documents, and **xml-schema-name** is the two-part SQL identifier of the annotated XML Schema.

```

>>-DECOMPOSE XML DOCUMENTS IN----'query'----XMLSCHEMA----->
                                     .-ALLOW NO ACCESS-.
>--xml-schema-name--+-----+--+-----+----->
                                     '-VALIDATE-' '-ALLOW ACCESS----'

>--+-----+--+-----+----->
   '-COMMITCOUNT--integer-' '-CONTINUE_ON_ERROR-'

>--+-----+----->
   '-MESSAGES--message-file-'

```

Figure 11.16 Syntax for the `DECOMPOSE XML DOCUMENTS` command

Figure 11.17 illustrates the execution of the `DECOMPOSE XML DOCUMENTS` command in the DB2 Command Line Processor.

```

DECOMPOSE XML DOCUMENTS IN 'SELECT cid, info FROM customer'
XMLSCHEMA db2admin.cust2xsd MESSAGES decomp_errors.xml ;

DB216001I The DECOMPOSE XML DOCUMENTS command successfully
decomposed all "100" documents.

```

Figure 11.17 Example of the `DECOMPOSE XML DOCUMENTS` command

If you don't specify a *message-file* then the error report is written to standard output. Figure 11.18 shows a sample error report. For each document that failed to shred, the error report shows the document identifier (`xdb:documentId`). This identifier is obtained from the first column that is produced by the SQL statement in the `DECOMPOSE XML DOCUMENTS` command. The error report also contains the DB2 error message for each document that failed. Figure 11.18 reveals that document 1002 contains an unexpected XML attribute called `status`, and that document 1005 contains an element or attribute value `abc` that is invalid because the XML Schema expected to find a value of type `xs:integer`. If you need more detailed information on why a document is not valid for a given XML Schema, use the stored procedure `XSR_GET_PARSING_DIAGNOSTICS`, which we discuss in section 17.6, *Diagnosing Validation and Parsing Errors*.

```
<?xml version='1.0' ?>
<xdb:errorReport
  xmlns:xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
  <xdb:document>
    <xdb:documentId>1002</xdb:documentId>
    <xdb:errorMsg>SQL16271N Unknown attribute "status" at or
      near line "1" in document "1002".</xdb:errorMsg>
  </xdb:document>
  <xdb:document>
    <xdb:documentId>1005</xdb:documentId>
    <xdb:errorMsg> SQL16267N An XML value "abc" at or near
      line "1" in document "1005" is not valid according to
      its declared XML schema type "xs:integer" or is outside
      the supported range of values for the XML schema type
    </xdb:errorMsg>
  </xdb:document>
</xdb:errorReport>
```

Figure 11.18 Sample error report from bulk decomp

11.4 SUMMARY

When you consider shredding XML documents into relational tables, remember that XML and relational data are based on fundamentally different data models. Relational tables are flat and unordered collections of rows with strictly typed columns, and each row in a table must have the same structure. One-to-many relationships are expressed by using multiple tables and join relationships between them. In contrast, XML documents tend to have a hierarchical and nested structure that can represent multiple one-to-many relationships in a single document. XML allows elements to be repeated any number of times, and XML Schemas can define hundreds or thousands of optional elements and attributes that may or may not exist in any given document. Due to these differences, shredding XML data to relational tables can be difficult, inefficient, and sometimes prohibitively complex.

If the structure of your XML data is of limited complexity such that it can easily be mapped to relational tables, and if your XML format is unlikely to change over time, then XML shredding can sometimes be useful to feed existing relational applications and reporting software.

DB2 offers two methods for shredding XML data. The first method uses SQL `INSERT` statements with the `XMLTABLE` function. One such `INSERT` statement is required for each target table and multiple statements can be combined in a stored procedure to avoid repetitive parsing of the same XML document. The shredding statements can include `XQuery` and `SQL` functions, joins to other tables, or references to `DB2` sequences. These features allow for customization and a high degree of flexibility in the shredding process, but require manual coding. The second approach for shredding XML data uses annotations in an XML Schema to define the mapping from XML to relational tables and columns. IBM Data Studio Developer provides a visual interface to create this mapping conveniently with little or no manual coding.

Index

Symbols

& (ampersand), escaping 88
* (asterisk) as wildcard character, 140, 594
@ (at sign) in XPath, 135
@* XPath wildcard, 140
, (comma) operator, construction of sequences, 154
\$ (dollar sign)
 in XML column references, 161
 XQuery variable names, 196
. (dot), current context in XPath, 151-153
// (double slash)
 in XPath, 141-142
 in XPath predicates, 146
!= (not equal) comparison operator, not() function versus, 150
% (percent sign) in wildcard searches, 583
.. (parent directory) in file system navigation, 133
.. (parent step) in XPath, 151-153

| (pipe) character
 union of sequences, 154
 as XPath union operator, 585
? (question mark) as wildcard character, 594
; (semicolon)
 in namespace declarations, 448
 in stored procedures, 549
' (single quotes), escaping, 571
/ (slash)
 in file system navigation, 133
 in XPath, 141
 in XPath predicates, 145
_ (underscore character) in wildcard searches, 583

A

abbreviated syntax in XPath, 157
access control, 9
access plans. *See* execution plans
ADD XMLSCHEMA command, 485
adjust-date-to-timezone function (XQuery), 226
ADMIN_EST_INLINE_LENGTH function, 45-47
ADMIN_IS_INLINED function, 44-45
ADO.NET data providers, list of, 631
aggregate functions, 278
aggregation. *See also* grouping
 XML construction with, 207-208
 of XML data, 233-239
 within and across documents, 236-237
 XMLTABLE function, 234-236
 with XMLAGG function (SQL/XML), 277-283
aggregation functions in XQuery, 218-220
ALTER INDEX command (DB2 Net Search Extender), 580
altering text indexes
 with DB2 Net Search Extender, 580
Altova XML tools, 656-658

ampersand (&), escaping 88
 AND operator, 149
 in full-text searches, 584
 annotated schema shredding, 306-318
 advantages/disadvantages of, 301
 annotating XML Schema, 306-310
 defining annotations
 in Data Studio Developer, 311
 registering annotated schemas, 311-312
 shredding multiple XML documents, 315-318
 shredding single XML documents, 312-315
 Annotated XSD Mapping Editor, 311
 APAR III4426, xxvi
 APARs, list of, 72-73
 APIs, 9
 application code page, 599
 application development, 609
 CLI applications, 636-639
 embedded SQL
 applications, 639-647
 C applications with, 645-647
 COBOL applications with, 640-642
 PL/1 applications with, 643-644
 for DB2 pureXML, 9
 host variables, 613-614
 Java applications, 615-631
 JDBC 3.0, XML support in, 615-619
 JDBC 4.0, example usage, 621-627
 JDBC 4.0, XML support in, 619-621
 pureQuery, 629-631
 XML data binding, 629
 XML documents, creating from application data, 627-628

.NET applications, 631-636
 ADO.NET data providers, list of, 631
 inserting XML data from, 635
 manipulating XML data in, 633-635
 querying XML data in, 632-633
 XML Schema and DTD handling, 636
 parameter markers, 613-614
 Perl applications, 650-651
 PHP applications, 647-649
 pureXML, benefits of, 610-613
 tools for
 Altova XML tools, 656-658
 IBM Data Studio Developer, 652-653, 655
 IBM Database Add-ins for Visual Studio, 656
 list of, 651
 <oXygen/>, 658-659
 Stylus Studio, 659
 application layer, avoiding parsing in, 610-612
 application-centric validation, 545
 applications (XML), best practices, 434-435
 arithmetic expressions, 190
 in XQuery, 212-214
 asterisk (*) as wildcard character, 140, 594
 atomic values (XQuery Data Model), 129
 attaching partitions, 57
 attribute axis, 157
 attribute constructors (XQuery), 290-292
 attribute expressions, XML construction with, 206
 attribute nodes, 29, 129
 attribute values versus, 136
 attribute values, attribute nodes versus, 136

attributes
 in path expressions, 135
 XPath wildcards for, 141
 attributes (objects), sparse, 13
 attributes (XML), 2-4. *See also* nodes
 constructing from relational data, 275-277
 converting to/from XML elements, 345-346
 elements versus, 15-19
 extracting value of, 557
 full names, 441, 443-444
 index creation and, 459
 indexing, 8
 inserting, defining position for, 336-337
 namespaces and, 440-441
 optional, 13
 renaming, 334-335
 updating with stored procedures, 554-555
 values, replacing, 327-328
 automatic updates for text indexes, 574-576
 axes in XPath, 157

B

BACKUP PENDING status, 111
 backward compatibility of XML Schema versions, 495-498
 base table row storage. *See* inlining
 BEFORE triggers, 523
 Bernoulli sampling, 419
 best practices for XML performance, 428-435
 between predicates, 431
 in XML queries, 254-256
 binary data as internally encoded, 618
 binary data types, 606

- binary SQL types, converting XML values to, 187-188
 - binding. *See* XML data binding
 - binding
 - BLOB data type, inserting XML documents 80
 - blobFromFile UDF, 81
 - blobsFromZipURL UDF, 81-82
 - blocking cursors, 435
 - BOM (Byte-Order Mark), 599
 - Boolean expressions, predicates versus, 146
 - Boolean functions in XQuery, 226
 - Boolean operators in full-text searches, 583-584
 - boost modifiers, 594
 - boundary whitespace, 90-91 preserving 91-93
 - bulk shredding of XML documents, 315-318
 - business data. *See* data business objects
 - data representation of, 12-13
 - storage of, 612
 - Byte-Order Mark (BOM), 599
- C**
- C applications with embedded SQL, 645-647
 - Call Level Interface. *See* CLI application development
 - cardinality of XML indexes, 363
 - Cartesian products, 240
 - case-insensitive XML queries, 252-253
 - cast expressions, 190
 - in XQuery, 208-212
 - castable XQuery expression, 211
 - casting. *See* converting
 - catalog tables (DB2 for z/OS), XML-related, 667-673
 - for XML indexes, 671-672
 - XML Schema Repository (XSR), 503-508, 672
 - for XML storage objects, 667-670
 - catalog views (DB2 for Linux, UNIX, and Windows), XML-related, 661-667
 - SYSCAT.COLUMNS, 661-662
 - SYSCAT.INDEXES, 663-664
 - SYSCAT.INDEXXML-PATTERNS, 664-666
 - SYSIBM.SYSXMLPATHS, 663
 - SYSIBM.SYSXML-STRINGS, 662-663
 - XML Schema Repository (XSR), 503-508, 667
 - change requests, response time for, 613
 - character data, as externally encoded, 619
 - character data types, blocking usage of, 606
 - character encoding. *See* XML encoding
 - character references, list of, 87
 - character type application variables, fetching non-Unicode data into, 603-604
 - check constraints, 8, 520-523
 - CHECK DATA utility, 69-70
 - CHECK INDEX utility, 66
 - CHECK PENDING status, 110
 - child axis, 157
 - Chinese characters in code page ISO-8859-1 (code page conversion example), 602-603
 - CLI (Call Level Interface) application development, 636-639
 - CLOB data type, inserting XML documents 80
 - CLP (Command Line Processor) DESCRIBE command, 84-85
 - escaping quotes in, 571
 - input parameters, text files as, 708
 - INSERT statements, 76-77
 - registering XML Schemas in, 484-486
 - retaining whitespace, 527
 - terminating characters, changing, 549
 - testing stored procedures, 555
 - truncated XML document display 83
 - viewing XML documents, 704-705
 - XML declarations, inserting 86
 - XML options list of, 706 usage examples, 706-707
 - coarse granularity of XML documents, 22
 - COBOL applications with embedded SQL, 640-642
 - code page conversions, 597
 - avoiding, 601
 - examples of, 602-605
 - with non-Unicode database code pages, 601-602
 - performance considerations, 434
 - code pages, selecting, 27
 - column references
 - dollar sign (\$) in, 161
 - in XMLQUERY function, 162-163
 - columns (XML) dropping, 40
 - generating from XML data, 165-166

- inserting constructed XML data into, 294-295
- comma (,) operator, construction of sequences, 154
- Command Line Processor. *See* CLP
- commands for full-text searches, list of, 594-595
- comment nodes, constructing, 290
- common table expressions, 282-283
- comparison expressions, 190
- comparison operations in predicates, 143
- comparison operators
 - numeric versus string comparison, 144
 - in XPath, 156-157
- compatibility. *See* backward compatibility
- compliance, data storage for, 94
- components. *See* schema documents
- compression
 - of XML data, 48-51
 - XML space management example, 54-57
- computed values
 - replacing values in XML documents with, 329-331
 - XML construction with, 202-204
- concat function (XQuery), 215-216
- concat() function, 155
- concatenation of text nodes, 30
- concurrency control, XML documents, 9
- conditional expressions, 190
 - XML construction with, 205
- conditional triggers, 524
- conditional XML element construction, 284-285
 - leading zeros in, 285-286
- configuring XML inlining, 43-47
- constraints on XML documents, 8, 520-523
- constructing XML data. *See* converting relational data to XML data; XML construction
- construction of sequences in XPath, 154-155
- constructor expressions, 190
- constructor functions. *See* publishing functions (SQL/XML)
- constructors (XQuery), 290-292
 - XML namespaces and, 462-463
- contains function (XQuery), 216-217, 587
- CONTAINS scalar function, 581-583
- content-centric XML documents, 567
- context (file system navigation), 133
- context nodes, 136, 139
- convert function, 229
- converting. *See also* shredding
 - relational data to XML data, 267
 - inserting in XML columns, 294-295
 - with SQL/XML publishing functions, 268-290
 - XML declarations for, 292-294
 - with XQuery constructors, 290-292
 - XML elements to/from XML attributes, 345-346
 - XML values to binary SQL types, 187-188
- COPY TABLESPACE utility, 66
- copying XML documents 86
- COPYTOCOPY utility, 66
- count() function, 155
- Creat Index Wizard, 366-367
- CREATE INDEX command (DB2 Text Search), 591-592
- CREATE INDEX command (DB2 Net Search Extender), 572-579
 - advanced options, 578-579
 - with automatic updates, 574-576
 - for parts of documents, 576-577
 - with specific storage paths, 573-574
- CREATE INDEX statement, 362-364
- current context in XPath, 151-153
- current directory (file system navigation), 133
- CURRENT IMPLICIT XMLPARSE OPTION register 93
- current-date function (XQuery), 225-226
- current-dateTime function (XQuery), 225
- current-time function (XQuery), 225
- cursors
 - loading from, 111
 - in stored procedures, 553-554
 - update cursors, modifying XML documents in, 350-351
- custom document models, full-text searches with, 585-586
- custom XML Schemas, industry standard XML Schemas versus, 474-476
- customer table (XML sample database), contents of, 710-712

D

- data, distinguishing from meta-
data, 19-21. *See also* rela-
tional data; XML data
- data binding (XML)
to Java objects, 629
pureQuery and, 631
- data exchange, metadata
for, 13
- data expansion/shrinkage (code
page conversion example),
605
- data function (XQuery), 221
- data loss due to XML
encoding, avoiding, 606
- data models. *See also* design
decisions
XML data, when to use,
11-13
XQuery 1.0 and XPath 2.0
Data Model, 126-131
sequence construction,
128-130
sequence input/output,
130-131
- data providers, list of, 631
- data storage. *See* storage
- Data Studio, support for DB2
pureXML, 9
- Data Studio Developer,
652-655
defining schema
annotations, 311
profiling stored
procedures, 556
- data types (SQL)
BLOB, inserting XML
documents, 80
CLOB, inserting XML
documents, 80
converting XML values
to binary SQL types,
187-188
DESCRIBE command,
84-85
index eligibility and,
374-375
type errors, avoiding in
XMLTABLE function,
168-169
XML, 7-9, 160
for XML indexes, 367-372
DATE, 369
DECFLOAT, 369
DOUBLE, 369
rejecting invalid
values, 371-372
selecting, 369-371
TIMESTAMP, 369
VARCHAR HASHED,
368-369
VARCHAR(n), 367-368
in XQuery, 208-212
- data types (Java), SQLXML, 9
- data() function, 134-135
- data-centric XML
documents, 567
- database code page,
non-Unicode database usage,
601-602
- database nodes. *See*
partitioned databases
- Database Partitioning Feature
(DPF), 59-60
- database utilities,
monitoring, 427-428
- database-centric
validation, 545
- databases
disabling
for DB2 Net Search
Extender, 572
for DB2 Text
Search, 591
enabling
for DB2 Net Search
Extender, 571-572
for DB2 Text Search,
590-591
XML sample database. *See*
XML sample database
DatabaseSpy, 658
DataDirect, 659
- date comparisons, string
comparisons versus,
210-211
- date functions in XQuery,
224-226
- DATE index data type, 369
- DB2 .NET Data Provider, 632
- DB2 Control Center
Creat Index Wizard,
366-367
support for DB2
pureXML, 9
viewing XML documents,
703-704
- DB2 Express-C, 196
- DB2 for Linux, UNIX, and
Windows, xxvi
explain facility, 396-409
exporting XML
documents, 98-106
importing XML
documents, 106-109
index implementation,
387-390
loading XML documents,
109-111
snapshot monitor,
424-427
statistics collection in,
418-419
validation in, DB2 for z/OS
versus, 543-544
XML compression, 48
XML index data types, 367
XML index statistics,
390-393
XML sample database,
creating, 709-710
XML Schemas in, 510-511
XML storage, 33-41
in DB2 9.7 release,
40-41
dropping XML
columns, 40
storage objects, types of,
33-35
table space page size,
36-39

- XML-related catalog views, 661-667
 - SYSCAT.COLUMNS, 661-662
 - SYSCAT.INDEXES, 663-664
 - SYSCAT.INDEXXML-PATTERNS, 664-666
 - SYSIBM.SYSXML-PATHS, 663
 - SYSIBM.SYSXML-STRINGS, 662-663
 - XML Schema Repository (XSR), 667
- DB2 9.1 for Linux, UNIX, and Windows, XML encoding, 597
- DB2 9.5 for Linux, UNIX, and Windows, XML encoding, 597
- DB2 9.7 for Linux, UNIX, and Windows, optimized XML storage format, 40-41
- DB2 for z/OS, xxvi
 - explain facility, 409-416
 - full-text searches in, 596
 - loading XML documents, 114-116
 - statistics collection in, 417-418
 - unloading XML documents, 111-114
 - updating XML documents in, 351-352
 - validation in, 540-544
 - DB2 for Linux, UNIX, and Windows versus, 543-544
 - for existing XML documents, 543
 - with INSERT statement, 541-542
 - with UPDATE statement, 542-543
 - XML compression, 48
 - XML encoding, 598
 - XML index data types, 367
- XML sample database, creating, 710
- XML Schemas in, 510-511
- XML storage, 60-73
 - limiting memory consumption, 71
 - multiple XML columns, 64
 - naming conventions, 64-65
 - offloading XML parsing, 72-73
 - storage objects, types of, 61-62
 - table space characteristics, 63
 - utilities for, 65-70
- XML-related catalog tables, 667-673
 - for XML indexes, 671-672
 - XML Schema Repository (XSR), 672
 - for XML storage objects, 667-670
- DB2 Net Search Extender administration commands, list of, 594-595
 - altering text indexes, 580
 - creating text indexes, 572-579
- DB2 Text Search versus, 568-570
 - disabling databases for, 572
 - enabling databases for, 571-572
 - performing full-text searches, 581-590
 - reorganizing text indexes, 579-580
 - updating text indexes, 579-580
- DB2 pureXML. *See* pureXML
- DB2 Text Search, 590
 - administration commands, list of, 594-595
 - creating text indexes, 591-592
 - DB2 Net Search Extender versus, 568-570
 - disabling databases for, 591
 - enabling databases for, 590-591
 - performing full-text searches, 592-594
- db2-fn:sqlquery function, 139, 166, 227, 229-230, 582
- db2-fn:xmlcolumn() function, 137, 166
- db2-fn:xmlcolumn-contains function, 592
- db2cat utility, 419-423
- db2exfmt command-line tool, 396-399
- db2look utility, XML documents and, 122
- db2move utility, XML documents and, 123
- DB2Xml class (.NET), 632-633
- DB2Xml object (JDBC 3.0), benefits of, 616
- DECFLOAT index data type, 369
- declarations (XML), 2, 599-600
 - in CLI applications, 638
 - for constructed XML data, 292-294
 - in embedded SQL applications, 639
 - handling documents with, 85-86
- declaring namespaces, 4
 - XML, 439-441
 - in SQL/XML, 451
 - in XML indexes, 456-460
 - in XMLTABLE function, 452-453
 - in XQuery, 448-450
 - XSLT, 356
- DECOMPOSE XML
- DOCUMENT command, 312

- DECOMPOSE XML
 - DOCUMENTS command, 317-318
- decomposing. *See* shredding
- dedicated directories, exporting XML documents to, 102-104
- default namespaces (XML), renaming nodes in, 467-468
- default tagging of relational data, 286-289
- default whitespace
 - preservation option, changing 93-94
- default XML namespaces, 442-444
- default XML Schemas, validation against with LOAD and IMPORT utilities, 532
- defining XML indexes, 362-367
- delete expression (XQuery), 333
- DELETE operator (execution plans), 401
- DELETE statement, 82-83
- delete triggers, 563
- deleting
 - XML documents, 82-83
 - XML nodes, 333-334
- delimited format files, 99
- descendant axis, 157
- descendant nodes, 141
- descendant-or-self axis, 157
- DESCRIBE command 84-85
- describing queries, 137
- design decisions, XML documents, 15-25, 428-429
 - elements versus attributes, 15-19
 - granularity, 22-24
 - hybrid storage, 24-25
 - performance, role of, 16
 - tags versus values, 19-21
- detaching partitions, 57
- DFETCH operator (execution plans), 413
- digital signatures, effect of stripping whitespace on, 78
- direct element construction, 171
- direct element/attribute constructors (XQuery), XML namespaces and, 462-463
- direct XML construction, 202
- directories, exporting XML documents to, 102-104
- directoryInfo UDF, 81
- disabling
 - annotated schemas for shredding, 312
 - databases
 - for DB2 Net Search Extender, 572
 - for DB2 Text Search, 591
- distinct-values function (XQuery), 221
- distribution keys, 60
- document ID index, 61
- document models, 576-577
 - custom document models, full-text searches with, 585-586
- document nodes, 29, 129
 - constructing, 294-295
- Document Object Model (DOM) parsers, 610
- Document Object Model fidelity, 94
- document trees (XML), 28-30
 - storage of, 30-33
- Document Type Definitions (DTDs), 501-502
- document validation. *See* validation
- document-centric XML documents. *See* content-centric XML
- documents (XML)
 - access control, 9
 - attribute values, replacing, 327-328
 - checking for validation, 534-535
 - constraints, 8
 - constructing
 - from multiple relational rows, 277-280
 - from multiple relational tables, 281-283
 - content-centric versus data-centric, 567
 - copying, 86
 - creating from Java application data, 627-628
 - db2look utility and, 122
 - db2move utility and, 123
 - deleting, 82-83
 - description of, 2-4
 - design decisions, 15-25, 428-429
 - elements versus attributes, 15-19
 - granularity, 22-24
 - hybrid storage, 24-25
 - performance, role of, 16
 - tags versus values, 19-21
 - document trees, 28-30
 - storage of, 30-33
 - element values, replacing, 326-327
 - elements/attributes, renaming, 334-335
 - escaping special characters, 87-89
 - exporting, 98-106
 - to dedicated directories, 102-104
 - fragments of documents, 104-105
 - to multiple files, 100-102
 - to single file, 98-100
 - with XML Schema information, 105-106
 - federating, 120-121
 - importing, 106-109
 - input files and, 107-108
 - performance tips, 108-109

indexing, 8
 inserting, 76-82
 from files, 79-82
 INSERT statement, 76-79
 loading
 in DB2 for Linux, UNIX, and Windows, 109-111
 in DB2 for z/OS, 114-116
 modifying
 in insert operations, 349-350
 in queries, 346-349
 in update cursors, 350-351
 with XQuery Update Facility, 324-326
 namespace declarations, 439-441
 namespace usage examples, 444-447
 nodes
 deleting, 333-334
 inserting, 335-340
 modifying multiple, 343-346
 repeating/missing, 340-343
 replacing, 331-332
 parameter markers, replacing values with, 328
 parsing, 9
 avoiding in application layer, 610-612
 publishing, 118-119
 queries on, 8-9
 removing validation, 540
 replacing, 322-324
 multiple values in, 328-329
 values with computed values, 329-331
 replicating, 118-119
 retaining invalid, 519-520
 retrieving, 83-85, 161-165

shredding, 10
 advantages/
 disadvantages of, 297-301
 with annotated schema shredding, 306-318
 with XMLTABLE function, 301-306
 splitting, 116-118
 storage. *See* XML storage
 transforming with XSLT, 352-358
 traversing, 197
 unloading, 111-114
 updating, 433
 in DB2 for z/OS, 351-352
 with UDFs, 559-561
 valid documents
 determining XML Schemas for, 538-540
 well-formed documents versus, 473
 validation. *See* validation
 viewing structure of, 703-705
 well-formed, 76
 whitespace, 89-94
 changing default preservation option 93-94
 preserving, 91-93
 types of, 90
 with XML declarations, handling, 85-86
 dollar sign (\$)
 in XML column references, 161
 XQuery variable names, 196
 DOM (Document Object Model) parsers, 610
 dot notation in XPath, 151-153
 DOUBLE index data type, 369
 double slash (//)
 in XPath, 141-142
 in XPath predicates, 146

DPF (Database Partitioning Feature), 59-60
 DROP XSROBJECT
 command, 492
 dropping
 check constraints, 522
 XML columns, 40
 DSNTIAUL command, 111-112
 DSN_XMLVALIDATE
 function, 541-543
 DTDs (Document Type Definitions), 501-502
 in .NET applications, handling, 636
 registering, 501
 dynamic XPath expressions, 185-186

E

EAV (Entity-Attribute-Value model). *See* Name/Value Pairs
 editing (Data Studio Developer)
 queries, 654
 XML Schemas, 653
 element constructors (XQuery), 290-292
 element nodes, 29-30, 129
 element values, returning without XML tags, 163-164
 elements (XML), 2-4. *See also* nodes
 attributes versus, 15-19
 constructing from relational data, 269-273
 conditional construction, 284-286
 empty, missing, NULL elements, 274-275
 converting to/from XML attributes, 345-346
 extracting repeating values, 557-558
 extracting value of, 557
 full names, 441-444

- indexing, 8
 - inserting, defining
 - position for, 335-336
 - leaf elements, 383
 - non-leaf elements, XML
 - indexes on, 383-384
 - optional elements
 - handling in XMLTABLE function, 167-168
 - schema flexibility of, 5
 - renaming, 334-335
 - repeating elements
 - numbering rows based on, 173-174
 - returning multiple, 174-176
 - returning with XMLQUERY function, 164-165
 - returning with XMLTABLE function, 169-173
 - schema flexibility of, 5
 - root elements, 28
 - updating with stored procedures, 554-555
 - values
 - replacing, 326-327
 - as text node
 - concatenations, 30
 - XPath wildcards for, 140
- embedded SQL application
- development, 639-647
 - C applications with, 645-647
 - COBOL applications with, 640-642
 - PL/1 applications with, 643-644
- embedding SQL in XQuery, 227-228
- empty elements (relational data), converting to XML data, 274-275
- “Empty on NULL” behavior, 274
- enabling
 - annotated schemas for shredding, 312
 - databases
 - for DB2 Net Search Extender, 571-572
 - for DB2 Text Search, 590-591
- encoding (XML). *See also* Unicode
- code page conversions
 - avoiding, 601
 - examples of, 602-605
 - code pages, selecting, 27
 - data loss, avoiding, 606
 - embedded SQL application development and, 639
 - external encoding, 599-601
 - internal encoding, 599-600
 - non-Unicode database usage, 601-602
 - overview, 597
- encoding declaration, 599
- enforcing validation
 - with check constraints, 520-523
 - with triggers, 523-525
- entities (XML), 87, 501
- entity references, list of 87
- Entity-Attribute-Value model (EAV). *See* Name/Value Pairs
- error codes
 - explained, 258-264
 - SQL0104N, 500
 - SQL0242N, 277
 - SQL0401N, 186
 - SQL0443N, 81
 - SQL0544N, 521
 - SQL0545N, 521
 - SQL0551N, 500
 - SQL1354N, 548
 - SQL1407N, 111
 - SQL16001N, 259
 - SQL16002N, 146, 259-260, 605
 - SQL16003N, 156, 169-170, 210, 213, 249, 260-261
 - SQL16005N, 261-262
 - SQL16011N, 263
 - SQL16015N, 262-263
 - SQL16061N, 144, 169, 211, 263-264, 551
 - SQL16075N, 136, 264
 - SQL16085N, 336, 339, 341-342
 - SQL16088N, 467
 - SQL16103N, 601
 - SQL16110N, 87
 - SQL16168N, 600
 - SQL16168N, 85
 - SQL16193N, 440
 - SQL16196N, 517
 - SQL16267N, 318
 - SQL16271N, 318
 - SQL20329N, 491
 - SQL20335N, 514
 - SQL20340N, 491
 - SQL20345N, 294, 337
 - SQL20353N, 186
 - SQL20412N, 604
 - SQL20429N, 606
 - SQL20432N, 498
 - SQLCODE -904, 71
 - SQLCODE 16002, 705
 - SQLSTATE 2200M, 519
- error handling
 - for registered XML Schemas, 490-491
 - in stored procedures, 551-553
 - for validation/parsing errors, 525-529
- escaping
 - ampersand (&), 88
 - less-than character (<), 87
 - quotes (’), 77, 88, 571
 - special characters, 87-89
- except operator, 155
- exchanging data. *See* data exchange

- executing
 - stored procedures, 547
 - triggers, 547
 - UDFs, 547
 - execution plans, 395-396
 - obtaining
 - with db2exfmt
 - command-line tool, 397-399
 - with SPUFI, 410-411
 - with Visual Explain tool, 400-401, 411-413
 - operators, list of, 401-403, 413-414
 - of stored procedures, 555-556
 - usage examples, 403-409, 414-416
 - existential semantics, 241, 254, 377
 - logical expressions
 - and, 149
 - in XPath, 147-148
 - existing XML documents, validating, 535-538
 - in DB2 for z/OS, 543
 - expanded names of XML elements/attributes, 441-444
 - explain facility
 - in DB2 for Linux, UNIX, and Windows, 396-409
 - db2exfmt command-line tool, 397-399
 - execution plan operators, 401-403
 - explain tables, 396-397
 - usage examples, 403-409
 - Visual Explain tool, 400-401
 - in DB2 for z/OS, 409-416
 - execution plan operators, 413-414
 - explain tables, 409-410
 - SPUFI, 410-411
 - usage examples, 414-416
 - Visual Explain tool, 411-413
 - explain tables
 - in DB2 for Linux, UNIX, and Windows, 396-397
 - in DB2 for z/OS, 409-410
 - EXPLAIN utility, 9
 - explaining stored procedure statements, 555-556
 - explicit serialization, 83, 294
 - EXPORT command, 98-106
 - exporting XML documents, 98-106
 - to dedicated directories, 102-104
 - fragments of documents, 104-105
 - to multiple files, 100-102
 - to single file, 98-100
 - with XML Schema information, 105-106
 - extensibility
 - in design decisions, 17
 - of XML, 1
 - eXtensible Markup Language. *See* XML
 - eXtensible Stylesheet Language Transformation. *See* XSLT
 - eXtensible Stylesheet Language. *See* XSL
 - external DTDs, 501
 - external encoding of character data, 619
 - external XML encoding, 599-601
 - extracting
 - repeating XML element values, 557-558
 - XML element/attribute values, 557
- F**
- f CLP option, 708
 - federating XML documents, 120-121
 - FETCH operator (execution plans), 401
 - file paths. *See* paths
 - file system navigation, 133
 - files, inserting XML documents from, 79-82
 - FILTER operator (execution plans), 401
 - filtering conditions on XMLQUERY function, 587
 - fine granularity of XML documents, 23
 - flexibility
 - in design decisions, 17
 - of XML Schema, 5-6
 - FLWOR expressions, 190-196
 - comparing with XPath and SQL/XML, 196-202
 - for and let clauses, compared, 193-194
 - for and let clauses, nested, 195-196
 - handling repeating/missing XML nodes, 342
 - join queries in, 247
 - in SQL/XML, 201-202
 - syntax of, 191-193
 - where and order by clauses, 194
 - for clause (FLWOR expressions)
 - let clause versus, 193-194
 - nested, 195-196
 - fragments of XML documents, exporting, 104-105
 - full names of XML elements/attributes, 441, 443-444
 - full-text searches
 - DB2 for z/OS, 596
 - DB2 Net Search Extender administration
 - commands, list of, 594-595

- altering text
 - indexes, 580
- creating text indexes, 572-579
- DB2 Text Search versus, 568-570
- disabling databases
 - for, 572
- enabling databases for, 571-572
- performing searches, 581-590
- reorganizing text
 - indexes, 579-580
- updating text indexes, 579-580
- DB2 Text Search, 590
 - administration
 - commands, list of, 594-595
 - creating text indexes, 591-592
 - disabling databases
 - for, 591
 - enabling databases for, 590-591
 - performing searches, 592-594
 - sample table for examples, 570-571
- fullselect (SQL), 555
- functions
 - XPath, 155
 - XQuery, 214-226
 - Boolean functions, 226
 - date and time functions, 224-226
 - namespace and node functions, 222-224
 - numeric and aggregation functions, 218-220
 - sequence functions, 220-222
 - string functions, 215-218
 - fuzzy searches, 586-587
- G**
 - general comparison operators
 - in XPath, 156
 - generated column, 557
 - GENROW operator (execution plans), 402
 - GET SNAPSHOT
 - command, 425
 - global declarations in XML Schemas, 478
 - global indexes, 58
 - global sequences,
 - performance optimization, 256-257
 - GRANT command, 499
 - granting XML Schema usage privileges, 499-500
 - granularity of XML
 - documents, 22-24, 428, 433
 - grouping XML data, 233-239.
 - See also* aggregation
 - in SQL/XML versus XQuery, 237-239
 - XMLTABLE function, 234-236
 - GUI for defining SQL/XML
 - publishing functions, 289-290
- H**
 - HADR (High Availability Disaster Recovery), 121
 - hashed indexes, 368
 - help. *See* technical support
 - hierarchical data, 12
 - hierarchical format, XML
 - document trees, 28-30
 - High Availability Disaster Recovery (HADR), 121
 - host variables, 183-184, 613-614
 - INSERT statements 78
 - performance considerations, 434
 - HSJOIN operator (execution plans), 402
 - HTML. *See* XML to HTML transformation
 - hybrid storage, 24-25, 299, 303-305
 - with stored procedures, 550-553
- I**
 - IBM Data Server Driver for JDBC and SQLJ. *See* JCC
 - IBM Data Studio Developer, 652-655
 - IBM Database Add-ins for Visual Studio, 656
 - IBM OmniFind Text Search Server for DB2
 - for z/OS, 596
 - IBM pureXML Technical Mastery Test, 675
 - ibm_db2 PHP extension, 647
 - identifiers for XML Schemas, 483, 516
 - ignoring stop words, 578
 - implicit parsing, 516
 - implicit serialization, 83, 294
 - implicit XML parsing, 354
 - IMPORT command, 106-109
 - input files and, 107-108
 - LOAD command
 - versus, 106
 - performance tips, 108-109
 - triggers and, 573
 - validating XML documents, 116, 530-534
 - against default XML Schemas, 532
 - against multiple XML Schemas, 530-532
 - against single XML Schema, 530-531
 - overriding XML Schema references, 532-534
 - schema location hints, 534

- importing
 - schema documents in XML Schemas, 479-482
 - XML documents, 106-109
 - input files and, 107-108
 - performance tips, 108-109
 - in-scope namespaces, 445, 455
 - in-scope-prefixes
 - function, 445
 - including schema documents in XML Schemas, 479-482
 - index directories, locating with work directories, 574
 - index eligibility, 373-374
 - data types and, 374-375
 - parent steps and, 385-386
 - text nodes and, 375-376
 - wildcards and, 376-377
 - XML namespaces and, 458-459
 - XMLQUERY and, 385
 - XQuery let and return clauses, 386-387
 - indexes
 - catalog tables for, 671-672
 - logical, 664-666
 - path indexes, 663
 - physical, 664-666
 - on range-partitioned tables, 58
 - regions indexes, 663
 - reorganization, 54
 - text indexes (DB2 Net Search Extender)
 - altering, 580
 - creating, 572-579
 - reorganizing, 579-580
 - updating, 579-580
 - user-defined XML, 664-666
 - on XML documents, 8
 - indexes (XML)
 - best practices, 432-433
 - cardinality of, 363
 - creating
 - with DB2 Control Center, 366-367
 - with XML namespaces, 456-460
 - data types for, 367-372
 - DATE, 369
 - DECFLOAT, 369
 - DOUBLE, 369
 - rejecting invalid values, 371-372
 - selecting, 369-371
 - TIMESTAMP, 369
 - VARCHAR HASHED, 368-369
 - VARCHAR(n), 367-368
 - DB2 for Linux, UNIX, and Windows implementation, 387-390
 - defining, 362-367
 - explain facility. *See* explain facility
 - join predicates and, 379-383
 - lean indexes, 365
 - logical and physical indexes, 389-390
 - on non-leaf elements, 383-384
 - parent steps and, 385-386
 - path indexes for, 387-389
 - query predicates and, 373-379
 - relational indexes
 - versus, 361
 - statistics, 390-393
 - for structural predicates, 377-379
 - unique indexes, 364-365
 - in XMLQUERY, 385
 - XQuery let and return clauses, 386-387
- industry standard XML Schemas, custom XML Schemas versus, 474-476
- InfoSphere Data Architect, 289-290
- InfoSphere Federation Server, 120
- inlining, 41-48, 429-430
 - benefits of, 47-48
 - drawbacks of, 48
 - monitoring and configuring, 43-47
 - viewing percentage of, 661-662
 - XML space management example, 54-57
- input, sequences as, 130-131
- input files, IMPORT command and, 107-108
- input parameters (CLP), text files as, 708
- input parameters (XML) in stored procedures, 548
- insert operations, modifying XML documents in, 349-350
- INSERT statement, 76-79
 - copying XML documents, 86
 - preserving whitespace, 92-93
 - validation, 514-517
 - in DB2 for z/OS, 541-542
 - XMLTABLE function, shredding XML documents with, 301-306
- insert triggers, 562-563
- inserting
 - constructed XML data into XML columns, 294-295
 - nodes in XML documents with namespaces, 468-469
 - XML data from .NET applications, 635
 - XML documents, 76-82
 - from files, 79-82
 - INSERT statement, 76-79
 - XML nodes, 335-340
- insignificant whitespace 90
- instances of the data model, 128

integer division in
XQuery, 214

integration, resources for information, 726

internal DTDs, 501

internal encoding
of binary data, 618
XML encoding, 599-600

intersect operator, 155

INTERSECT operator
(execution plans), 413

invalid XML documents,
retaining, 519-520

invalid XML index data type
values, rejecting, 371-372

ISO-8859-1, Chinese
characters in (code page
conversion example),
602-603

items (XQuery Data
Model), 129

J

Japanese literal values in non-Unicode database
(code page conversion
example), 605

Java application
development, 615-631
JDBC 3.0, XML support in,
615-619
JDBC 4.0, 9
example usage,
621-627
XML support in,
619-621
pureQuery, 629-631
XML data binding, 629
XML documents, creating
from application data,
627-628

Java applications
inserting XML documents
from, 78-79
registering XML Schemas
from, 488

JCC (Java Common
Client), 615

JDBC
registering XML Schemas
with, 488
support for, 615

JDBC 3.0, XML support in,
615-619

JDBC 4.0, 9
example usage, 621-627
XML support in, 619-621

join predicates, XML indexes
and, 379-383

join queries, 239
outer joins, 250-252
in SQL/XML, 242-247
XML-to-relational joins,
248-250
in XQuery, 240-242

joins
best practices, 431
XML versus relational data,
7, 241

K

key cardinalities in XML
indexes, 390

Key-Value Pairs (KVP). *See*
Name/Value Pairs

known whitespace 90

Korean character code page
conversion example, 605

KVP (Key-Value Pairs). *See*
Name/Value Pairs

L

last function (XQuery), 222

last() function, 153

leading zeros in conditional
XML element construction,
285-286

leaf elements, 383

lean XML indexes, 365

left outer joins, 250

legacy functions
(SQL/XML), 290

less-than character (<),
escaping, 87

let clause (FLWOR expressions)
for clause versus, 193-194
nested, 195-196

let clause (XQuery), index
eligibility and, 386-387

Linux. *See* DB2 for Linux,
UNIX, and Windows

list tablespaces command, 51

LIST UTILITIES
command, 427

LISTDEF utility, 69

LOAD command, 109-111,
114-116
IMPORT command
versus, 106
triggers and, 573
validating XML
documents, 116, 530-534
against default XML
Schemas, 532
against multiple XML
Schemas, 530-532
against single XML
Schema, 530-531
overriding XML Schema
references, 532-534
schema location
hints, 534

LOAD QUERY command, 428

loading XML documents
in DB2 for Linux, UNIX,
and Windows, 109-111
in DB2 for z/OS, 114-116

LOB storage
pureXML storage versus,
10-11
for XML data, 10

local declarations in XML
Schemas, 478

local indexes, 58

local names of XML
elements/attributes,
441-444

local-name function
(XQuery), 223

locale-aware Unicode
collations, 252
locators, 577
locking XML documents, 9
logical expressions, 190
in XPath, 148-151
logical indexes, 664-666
XML indexes, 389-390
loops in stored procedures,
553-554

M

manipulating XML data. *See*
XML manipulation
MapForce, 657
mapping
path indexes for XML
indexes, 387-389
paths to pathIDs, 663
relational data to XML data,
GUI-based
definition, 289-290
tag names to stringIDs,
31-33
XML data to relational data.
See annotated schema
shredding
XML Schema pairs, 533
XML tags to
stringIDs, 662
marshalling, 629
MDC (multidimensional
clustering), 58-59
medium granularity of XML
documents, 22
memory consumption,
limiting in DB2 for
z/OS, 71
metadata
distinguishing from data,
19-21
for data exchange, 13
missing elements (relational
data), converting to XML
data, 274-275. *See also*
optional elements

missing XML nodes,
handling, 340-343
mixed content, 143
in XML document trees,
29-30
modifying. *See also* updating
multiple XML nodes,
343-346
XML documents
in insert operations,
349-350
in queries, 346-349
in update cursors,
350-351
with XQuery Update
Facility, 324-326
monitoring
performance, 424
of database utilities,
427-428
with snapshot monitor,
424-427
XML inlining, 43-47
moving. *See* exporting; import-
ing; inserting; loading;
unloading
multidimensional clustering
(MDC), 58-59
multiple documents,
constructing from queries,
253-254
multiple files, exporting XML
documents to, 100-102
multiple for/let clauses
(FLWOR expressions),
195-196
multiple namespaces in XML
documents, 440-441
multiple nesting levels, XML
construction with, 206-207
multiple node values in XML
documents, replacing,
328-329
multiple relational rows,
constructing XML
documents from, 277-280

multiple relational tables,
constructing XML
documents from, 281-283
multiple repeating elements,
returning, 174-176
multiple schema documents in
XML Schemas, 479-482
multiple table spaces,
performance and, 37
multiple XML columns
in DB2 for z/OS, 64
populating, 108
multiple XML documents,
shredding, 315-318
multiple XML namespaces,
querying XML documents
with, 454-456
multiple XML nodes,
modifying, 343-346
multiple XML Schemas,
validation
with LOAD and IMPORT
utilities, 530-532
with triggers, 524

N

Name/Value Pairs (NVP),
20-21
namespace functions
in XQuery, 222-224
namespaces (XML), 437-439
constructing XML data
with, 460-463
creating indexes with,
456-460
declaring, 4, 439-441
for XSLT, 356
default, 442-444
full-text searches and,
588-590
querying XML data with,
447-456
updating XML data with,
463-469
usage examples, 444-447

- XML indexes and, 432
 - in XML sample database tables, 710
 - naming conventions
 - XML storage in DB2 for z/OS, 64-65
 - XML tags, 4
 - nested for/let clauses (FLWOR expressions), 195-196
 - nested predicates, 150
 - nested XQuery functions, 217
 - nesting
 - SQL and XQuery, 257-258
 - XML tags, 3
 - XMLELEMENT functions, 270-273
 - nesting levels, XML
 - construction with, 206-207
 - .NET application
 - development, 631-636
 - ADO.NET data providers, list of, 631
 - inserting XML data from, 635
 - manipulating XML data in, 633-635
 - querying XML data in, 632-633
 - XML Schema and DTD handling, 636
 - Net Search Extender. *See* DB2 Net Search Extender
 - node functions in XQuery, 222-224
 - node tests, 133
 - NodeID index, 62
 - nodes. *See also* partitioned databases
 - attribute nodes, attribute values versus, 136
 - context nodes, 136, 139
 - descendant nodes, 141
 - document nodes, constructing, 294-295
 - inserting/replacing in XML documents with namespaces, 468-469
 - renaming
 - in XML documents with default namespaces, 467-468
 - in XML documents with prefixed namespaces, 465-467
 - text nodes, index
 - eligibility and, 375-376
 - types of, 28
 - values, replacing
 - with computed values, 329-331
 - multiple values, 328-329
 - with parameter markers, 328
 - in XML documents
 - deleting, 333-334
 - inserting, 335-340
 - modifying multiple, 343-346
 - repeating/missing, 340-343
 - replacing, 331-332
 - XQuery Data Model, 129
 - non-leaf elements, 30, 134
 - XML indexes on, 383-384
 - non-Unicode databases
 - avoiding data loss in, 606
 - for XML data management, 601-602
 - normalization, 7
 - of business objects, 12
 - not equal (!=) comparison operator, not() function versus, 150
 - NOT operator in full-text searches, 584
 - not() function, 148, 150
 - not equal (!=) comparison operator versus, 150
 - NSE. *See* DB2 Net Search Extender
 - NULL, setting XML columns to, 82
 - NULL elements (relational data), converting to XML data, 274-275
 - “NULL on NULL” behavior, 274
 - numbering rows based on repeating elements, 173-174
 - NUMBEROFMATCHES scalar function, 581-583
 - numeric comparisons, string comparisons versus, 144, 211-212
 - numeric functions in XQuery, 218-220
 - NVP (Name/Value Pairs), 20-21
- O**
- octets, 188
 - offloading XML parsing in DB2 for z/OS, 72-73
 - OmniFind Text Search Server for DB2 for z/OS, 596
 - one-to-many relationships, XML elements, 3
 - online table moves, 40
 - operators (for execution plans), 395
 - list of, 401-403, 413-414
 - usage examples, 403-409, 414-416
 - optimization of queries, 253-258
 - between predicates, 254-256
 - large global sequences, 256-257
 - nesting SQL and XQuery, 257-258
 - single versus multiple document construction, 253-254
 - optional attributes (XML), 13

- optional elements (XML)
 - handling in XMLTABLE function, 167-168
 - schema flexibility of, 5
 - OR operator, 149-150
 - in full-text searches, 583-584
 - order by clause (FLWOR expressions), 194
 - ordering result sets by XML values, 186-187
 - outer joins, 250-252
 - output, sequences as, 130-131
 - overriding XML Schema references in LOAD and IMPORT utilities, 532-534
 - <oXygen/>, 658-659
- P**
- page size
 - of table spaces, 36-39
 - for XML storage, 429
 - page-level sampling, 419
 - pairs (XML Schemas), mapping, 533
 - parameter markers, 183-184, 613-614
 - INSERT statements 78
 - performance considerations, 434
 - replacing values with, 328
 - parent axis, 157
 - parent of current directory (file system navigation), 133
 - parent steps
 - index eligibility and, 385-386
 - in XPath, 151-153
 - parsing, 30
 - avoiding in application layer, 610-612
 - error handling, 525-529
 - implicit parsing, 516
 - pureQuery and, 631
 - valid versus well-formed XML documents, 473
 - XML documents, 9
 - offloading in DB2 for z/OS, 72-73
 - performance considerations, 434
 - with special characters 88
 - partial shredding, 299
 - partition elimination, 57
 - PARTITION operator (execution plans), 413
 - partitioned databases, 59-60
 - partitioning, range, 57-58
 - path expressions, 190
 - path indexes, 35, 58, 663
 - for XML indexes, 387-389
 - pathIDs, mapping to paths, 663
 - paths
 - in IMPORT command, 107
 - mapping to pathIDs, 663
 - storage paths for text indexes, 573-574
 - pdo_ibm PHP extension, 647
 - percent sign (%) in wildcard searches, 583
 - performance
 - best practices, 428-435
 - explain facility
 - in DB2 for Linux, UNNIX, and Windows, 396-409
 - in DB2 for z/OS, 409-416
 - importing XML documents, 108-109
 - LOAD command, 110
 - mapping tag names to stringIDs, 32
 - monitoring, 424
 - of database utilities, 427-428
 - with snapshot monitor, 424-427
 - multiple table spaces and, 37
 - partition elimination, 57
 - query optimization, 253-258
 - between predicates, 254-256
 - large global sequences, 256-257
 - nesting SQL and XQuery, 257-258
 - single versus multiple document construction, 253-254
 - role in design decisions, 16
 - text indexes and, 574
 - of XSLT processing, 353
 - Perl application development, 650-651
 - PHP application development, 647-649
 - physical indexes, 664-666
 - physical XML indexes, 389-390
 - pipe (|) character
 - union of sequences, 154
 - as XPath union operator, 585
 - PL/1 applications with embedded SQL, 643-644
 - plain SQL (XML data queries), 127
 - position() function, 154
 - positional predicates in XPath, 153-154
 - positional relationships in search conditions, 588
 - positioning
 - inserted XML attributes, 336-337
 - inserted XML elements, 335-336
 - predicates
 - in FLWOR expressions, 192
 - join predicates, XML indexes and, 379-383
 - query examples of, 198-199
 - query predicates, XML indexes and, 373-379
 - structural predicates, XML indexes for, 377-379

- usage with SQL/XML, 177-181
 - common mistakes, 181-182
 - XML construction with, 204-205
 - in XPath, 142-146
 - dot notation, 151-153
 - existential semantics, 147-148
 - logical expressions, 148-151
 - positional predicates, 153-154
 - prefixed namespaces (XML), 438-439
 - mixing with default XML namespaces, 442
 - renaming nodes in, 465-467
 - PreparedStatement interface (JDBC 3.0), 618
 - preserving whitespace, 91-93
 - changing default, 93-94
 - during import, 108
 - validation and, 517
 - pretty print, CLP option for, 707
 - primary schema documents, 481
 - privileges for XML Schema usage, granting/revoking, 499-500
 - processing instruction nodes, constructing, 290
 - product table (XML sample database), contents of, 712-713
 - profiling stored procedures, 556
 - prototyping, XML flexibility for, 612-613
 - proximity searches, 586
 - publishing functions (SQL/XML), 160, 268-290
 - combining with XQuery constructors, 292
 - empty, missing, NULL elements, 274-275
 - GUI-based definition, 289-290
 - legacy functions, 290
 - list of, 268
 - XML namespaces and, 460-462
 - XMLAGG, 277-283
 - XMLAGG, XMLCONCAT, XMLFOREST compared, 284
 - XMLATTRIBUTES, 275-277
 - XMLCOMMENT, 290
 - XMLCONCAT, 270
 - XMLELEMENT, 269-273
 - XMLFOREST, 272-273
 - XMLGROUP, 286-289
 - XMLPI, 290
 - XMLROW, 286-289
 - XMLTEXT, 290
 - publishing XML documents, 118-119
 - purchaseorder table (XML sample database), contents of, 713-714
 - pureQuery, 629-631
 - pureXML
 - for application development, benefits of, 610-613
 - functionality of, xxiii-xxiv, 7-10
 - quiz, 675-702
 - XML data storage methods versus, 10-11
- Q**
- Q Apply, 119
 - q CLP option, 527, 706
 - Q replication, 119
 - queries. *See also* querying XML data
 - against XSR (XML Schema Repository), 508-510
 - editing in Data Studio Developer, 654
 - query predicates, XML indexes and, 373-379
 - querying XML data, 8-9
 - best practices, 430-432
 - case-insensitive queries, 252-253
 - error codes, 258-264
 - execution plans, 395-396
 - explain facility
 - in DB2 for Linux, UNIX, and Windows, 396-409
 - in DB2 for z/OS, 409-416
 - grouping and aggregation, 233-239
 - in SQL/XML versus XQuery, 237-239
 - within and across documents, 236-237
 - XMLTABLE function, 234-236
 - join queries, 239
 - in SQL/XML, 242-247
 - in XQuery, 240-242
 - outer joins, 250-252
 - XML-to-relational joins, 248-250
 - methods of, 126-127
 - in .NET applications, 632-633
 - overview, 126-128
 - performance optimization, 253-258
 - between predicates, 254-256
 - large global sequences, 256-257
 - nesting SQL and XQuery, 257-258
 - single versus multiple document construction, 253-254

sample data for examples, 131-132

SQL/XML, 159-160

- converting XML values to binary SQL types, 187-188
- dynamic XPath expressions, 185-186
- host variables, 183-184
- namespace declarations, 451
- ordering result sets, 186-187
- overview, 160
- parameter markers, 183-184
- performance considerations, 434
- retrieving XML documents, 161-165
- retrieving XML values in relational format, 165-176
- XPath predicate usage, 177-182

with XML namespaces, 447-456

XPath

- axes, 157
- comparison operators, 156-157
- construction of sequences, 154-155
- data() function, 134-135
- dot notation, 151-153
- double slash (//), 141-142
- empty results, reasons for, 134
- executing in DB2, 137-140
- existential semantics, 147-148
- file system navigation analogy, 133
- functions, 155

- logical expressions, 148-151
- node tests, 133
- positional predicates, 153-154
- predicates, 142-146
- simple query examples, 133-136
- slash (/), 141
- string() function, 135
- text() node test, 134
- unabbreviated syntax, 157
- union of sequences, 154-155
- wildcards, 140-141

XQuery

- arithmetic expressions, 212-214
- attribute expressions in XML construction, 206
- comparing FLWOR expressions, XPath, SQL/XML, 196-202
- computed value XML construction, 202-204
- conditional expressions in XML construction, 205
- data types, cast expressions, type errors, 208-212
- direct XML construction, 202
- embedding SQL in, 227-228
- FLWOR expressions, 191-196
- functions, 214-226
- modifying XML documents in, 346-349
- multiple nesting levels in XML construction, 206-207

- namespace and node functions, 445
- namespace declarations, 448-450
- overview, 190
- predicates in XML construction, 204-205
- SQL functions and UDFs in, 229-230
- XML aggregation in XML construction, 207-208

XQuery Data Model, 128-131

- sequence construction, 128-130
- sequence input/output, 130-131

question mark (?) as wildcard character, 594

questions. *See* technical support

quiz on pureXML, 675-702

quotes ('), escaping, 77, 88, 571

R

- range partitioning, 57-58
- rapid prototyping, 612-613
- RDA (Rational Data Architect), 289
- REAL TIME STATISTICS utility, 66
- REC2XML function (SQL/XML), 290
- RECOVER INDEX utility, 66
- RECOVER TABLESPACE utility, 66
- referencing
 - XML columns. *See* XML column references
 - XML Schemas, 484
- referential integrity of XML documents, 8
- regions, 34-35
- page size and, 36

- regions indexes, 34, 58, 663
- REGISTER XMLSCHEMA
 - command, 311, 484
- registering
 - annotated schemas, 311-312
 - DTDs, 501
 - XML Schemas, 483-491
 - in CLP (command-line processor), 484-486
 - error handling for, 490-491
 - identifiers, 483
 - with JDBC, 488
 - with shared schema documents, 489-490
 - steps in, 483
 - with stored procedures, 486-487
- relational data
 - converting to XML data, 267
 - inserting in XML columns, 294-295
 - with SQL/XML publishing functions, 268-290
 - XML declarations for, 292-294
 - with XQuery constructors, 290-292
- converting XML documents to
 - advantages/disadvantages, 297-301
 - with annotated schema shredding, 306-318
 - with XMLTABLE function, 301-306
- generating Java classes from, 629-631
- hybrid storage, 24-25
- XML versus, 4-7
 - when to use XML data, 11-13
- XML-to-relational joins, 248-250
- relational format, retrieving XML values in, 165-176
- relational indexes, XML indexes versus, 361
- relational joins, XML joins versus, 241
- relational views over XML data, 305-306
- relationships, one-to-many, 3
- removing. *See also* deleting; stripping
 - validation from XML documents, 540
 - XML Schemas from XSR, 492-493
- renaming nodes
 - in XML documents with default namespaces, 467-468
 - in XML documents with prefixed namespaces, 465-467
 - XML elements/attributes, 334-335
- REORG command, 53-54, 68-69
- reorganizing
 - text indexes with DB2 Net Search Extender, 579-580
 - XML indexes, 433
 - XML space management example, 54-57
 - XML table data, 53-54, 68-69
- repeating elements (XML), 3
 - extracting values of, 557-558
 - numbering rows based on, 173-174
 - returning
 - multiple elements, 174-176
 - with XMLQUERY function, 164-165
 - with XMLTABLE function, 169-173
 - schema flexibility of, 5
- repeating XML nodes, handling, 340-343
- replace expression (XQuery), 331
- replacing. *See also* updating nodes in XML documents
 - with namespaces, 468-469
 - XML attribute values, 327-328
 - XML documents, 322-324
 - XML element values, 326-327
 - XML node values
 - with computed values, 329-331
 - multiple node values, 328-329
 - with parameter markers, 328
 - XML nodes, 331-332
- replicating XML documents, 118-119
- REPORT TABLESPACESET utility, 67-68
- reserved characters. *See* special characters
- RESET MONITOR command, 425
- resources for information, 717-726
 - on Altova XML tools, 658
- response time for change requests, 613
- result set cardinalities, 200-201
- result sets, ordering by XML values, 186-187
- ResultSet interface (JDBC 3.0), 615
- retaining
 - invalid XML documents, 519-520
 - whitespaces in CLP, 527

- retrieving
 - XML documents, 83-85, 161-165
 - XML values in relational format, 165-176
 - return clause (XQuery), index eligibility and, 386-387
 - RETURN operator (execution plans), 402
 - returning element values
 - without XML tags, 163-164
 - revised XML Schemas. *See* XML Schema evolution
 - REVOKE command, 500
 - revoking XML Schema usage privileges, 499-500
 - RIDSCN operator (execution plans), 402
 - right outer joins, 251
 - root elements (XML), 4, 28
 - row-level sampling, 419
 - rows
 - generating from XML data, 165-166
 - numbering based on repeating elements, 173-174
 - RPD operator (execution plans), 402
 - RUNSTATS INDEX utility, 67
 - RUNSTATS TABLESPACE utility, 67
 - RUNSTATS utility, 9, 50, 417, 666
 - in DB2 for Linux, UNIX, and Windows, 418-419
 - in DB2 for z/OS, 417-418
- S**
- sample database. *See* XML sample database
 - sampling in statistics
 - collection, 419
 - SAX (Simple API for XML)
 - parsers, 611
 - scalar functions, 162, 200, 557
 - for full-text searches, 581-583
 - scalar subselects, 282
 - schema documents, 476
 - multiple schema documents in XML Schemas, 479-482
 - sharing between XML Schemas, 489-490
 - schema location hints in LOAD and IMPORT
 - utilities, 534
 - schema names, comparison with XML namespaces, 438
 - schema validation. *See* validation
 - schemas (XML). *See also* XML Schema
 - best practices, 434
 - volatility of, 12
 - SCORE scalar function, 581-583
 - search conditions. *See also* predicates
 - parts of, 582
 - positional relationships in, 588
 - search term (in search conditions), 582
 - searches. *See* full-text searches
 - section (in search conditions), 582
 - SELECT statement, retrieving XML documents, 83-85
 - selecting
 - code pages, 27
 - XML index data types, 369-371
 - self axis, 157
 - self-describing data format, XML as, 19
 - self-joins, 228
 - semicolon (;)
 - in namespace declarations, 448
 - in stored procedures, 549
 - sequence constructors, 175
 - sequence expressions, 190
 - sequence functions in XQuery, 220-222
 - sequences, 550
 - constructing, 128-130
 - global sequences, performance optimization, 256-257
 - as input/output, 130-131
 - in XPath, 154-155
 - serialization, 30, 83, 138
 - SET INTEGRITY
 - command, 110
 - SET INTEGRITY PENDING
 - status, 110
 - sharing schema documents
 - between XML Schemas, 489-490
 - SHIP operator (execution plans), 402
 - shredding
 - pureXML storage versus, 10-11
 - XML data with UDFs, 558-559
 - XML documents, 10
 - advantages/disadvantages of, 297-301
 - with annotated schema shredding, 306-318
 - with XMLTABLE function, 301-306
 - sibling branches, search conditions on, 588
 - significant whitespace 90
 - Simple API for XML (SAX)
 - parsers, 611
 - SimpleXML PHP
 - extension, 647
 - single documents,
 - constructing from queries, 253-254
 - single quotes ('), escaping, 77, 88, 571
 - size. *See* granularity

- slash (/)
 - in file system
 - navigation, 133
 - in XPath, 141
 - in XPath predicates, 145
- snapshot monitor, 424-427
- snapshot semantics, 343-345
- SNAPTAB_REORG
 - administrative view, 428
- SNAPUTIL administrative view, 427
- SNAPUTIL_PROGRESS
 - administrative view, 427
- SORT operator (execution plans), 402
- sparse attributes, 13
- special characters, escaping, 87-89
- splitting XML documents, 116-118
- SPUFI
 - execution plans, obtaining, 410-411
 - viewing XML documents, 705
- SQL. *See also* SQL/XML
 - embedding in XQuery, 127, 227-228
 - nesting with XQuery, 257-258
 - scalar functions for full-text searches, 581-583
 - stored procedures. *See* stored procedures for XML data queries, 127
- SQL functions in XQuery, 229-230
- SQL statements, embedding XPath/XQuery in, 127
- SQL/XML, 8, 159-160
 - comparing with FLWOR expressions and XPath, 196-202
 - converting XML values to binary SQL types, 187-188
 - dynamic XPath expressions, 185-186
 - FLWOR expressions in, 201-202
 - grouping queries in, XQuery versus, 237-239
 - host variables, 183-184
 - INSERT statement, validation on, 514-517
 - join queries, XML-to-XML joins, 242-247
 - namespace declarations, 451
 - ordering result sets, 186-187
 - overview, 160
 - parameter markers, 183-184
 - performance considerations, 434
 - publishing functions, 268-290
 - combining with XQuery constructors, 292
 - empty, missing, NULL elements, 274-275
 - GUI-based definition, 289-290
 - legacy functions, 290
 - list of, 268
 - XML namespaces and, 460-462
 - XMLAGG, 277-283
 - XMLAGG, XMLCONCAT, XMLFOREST compared, 284
 - XMLATTRIBUTES, 275-277
 - XMLCOMMENT, 290
 - XMLCONCAT, 270
 - XMLELEMENT, 269-273
 - XMLFOREST, 272-273
 - XMLGROUP, 286-289
 - XMLPI, 290
 - XMLROW, 286-289
 - XMLTEXT, 290
 - UPDATE statement, validation on, 518-519
 - XML aggregation, XML construction with, 207-208
 - XML documents, retrieving, 161-165
 - XML values, retrieving in relational format, 165-176
 - XPath and XQuery versus, 201
 - XPath predicate usage, 177-181
 - common mistakes, 181-182
- SQL0104N error code, 500
- SQL0242N error code, 277
- SQL0401N error code, 186
- SQL0443N error code, 81
- SQL0544N error code, 521
- SQL0545N error code, 521
- SQL0551N error code, 500
- SQL1354N error code, 548
- SQL1407N error code, 111
- SQL16001N error code, 259
- SQL16002N error code, 146, 259-260, 605
- SQL16003N error code, 156, 169-170, 210, 213, 249, 260-261
- SQL16005N error code, 261-262
- SQL16011N error code, 263
- SQL16015N error code, 262-263
- SQL16061N error code, 144, 169, 211, 263-264, 551
- SQL16075N error code, 136, 264
- SQL16085N error code, 336, 339, 341-342
- SQL16088N error code, 467
- SQL16103N error code, 601
- SQL16110N error code, 87
- SQL16168N error code, 600
- SQL16168N error code, 85

- SQL16193N error code, 440
- SQL16196N error code, 517
- SQL16267N error code, 318
- SQL16271N error code, 318
- SQL20329N error code, 491
- SQL20335N error code, 514
- SQL20340N error code, 491
- SQL20345N error code, 294, 337
- SQL20353N error code, 186
- SQL20412N error code, 604
- SQL20429N error code, 606
- SQL20432N error code, 498
- SQLCODE -904 error code, 71
- SQLCODE 16002 error code, 705
- SQLSTATE 2200M error code, 519
- SQLXML interface (JDBC 4.0), 619-621
- SQLXML Java data type, 9
- star. *See* asterisk (*)
- starts-with function (XQuery), 218
- statement heap, size of, 432
- statistics
 - db2cat utility, 419-423
 - RUNSTATS utility, 417
 - in DB2 for Linux, UNIX, and Windows, 418-419
 - in DB2 for z/OS, 417-418
 - for XML indexes, 390-393
- StAX (Streaming API for XML) parsers, 611
- stemming in full-text searches, 586
- steps (file system navigation), 133
- stop words, ignoring, 578
- storage. *See also* data storage
 - of business objects, 612
 - for compliance 94
 - hybrid XML data storage with stored procedures, 550-553
 - pureXML versus
 - alternative XML storage methods, 10-11
 - of XML document trees, 30-33
 - XML storage, 429-430
 - in DB2 for Linux, UNIX, and Windows, 33-41
 - in DB2 for z/OS, 60-73
 - inlining, 41-48
 - MDC (multidimensional clustering), 58-59
 - partitioned databases, 59-60
 - range partitioning, 57-58
 - space consumption of, 51-53
 - space management example, 54-57
- storage objects
 - catalog tables for, 667-670
 - in DB2 for Linux, UNIX, and Windows, types of, 33-35
 - in DB2 for z/OS, types of, 61-62
- storage paths for text indexes, 573-574
- stored procedures, 548-556
 - benefits of, 547
 - for dynamic XPath expressions, 185-186
 - executing, 547
 - for hybrid XML data storage, 550-553
 - loops and cursors, 553-554
 - registering XML Schemas with, 486-487
 - retaining invalid XML documents, 519-520
 - for shredding XML documents, 313-315
 - testing, 555-556
 - updating XML elements/attributes, 554-555
- Streaming API for XML (StAX) parsers, 611
- string comparisons
 - case-insensitivity, 252-253
 - date comparisons versus, 210-211
 - numeric comparisons versus, 144, 211-212
- string functions in XQuery, 215-218
- string() function, 135
- string-join function, 171, 216
- stringIDs, mapping
 - tag names to, 31-33
 - to XML tags, 662
- stripping whitespace, 78
 - changing default, 93-94
- structural predicates, 147, 153
 - XML indexes for, 377-379
- structure of XML documents, viewing, 703-705
- style sheets. *See* XSLT
- StyleVision, 657
- Stylus Studio, 659
- subselects, 282
- substring-after function (XQuery), 217
- synchronous index
 - maintenance, 361
 - syntax of FLWOR expressions, 191-193
- SYSCAT.COLUMNS catalog view, 661-662
- SYSCAT.INDEXES catalog view, 663-664
- SYSCAT.INDEXXML-PATTERNS catalog view, 664-666
- SYSCAT.XDBMAPGRAPHS catalog view, 504, 508, 667
- SYSCAT.XDBMAP-SHREDTREES catalog view, 504, 508, 667
- SYSCAT.XSROBJECTAUTH catalog view, 504, 507, 667

SYSCAT.XSROBJECT-COMPONENTS catalog view, 504, 506, 667
 SYSCAT.XSROBJECTDEP catalog view, 504, 507, 667
 SYSCAT.XSROBJECT-HIERARCHIES catalog view, 504, 506, 667
 SYSCAT.XSROBJECTS catalog view, 503, 505, 667
 SYSIBM.SYSINDEXES catalog table, 671
 SYSIBM.SYSKEYTARGETS catalog table, 671-672
 SYSIBM.SYSTABLES catalog table, 668-670
 SYSIBM.SYSTABLESPACE catalog table, 670
 SYSIBM.SYSXMLPATHS catalog view, 663
 SYSIBM.SYSXMLRELS catalog table, 667-668
 SYSIBM.SYSXMLSTRINGS catalog table, 668
 SYSIBM.SYSXMLSTRINGS catalog view, 662-663
 SYSIBMTS.TSCOLLECTIONNAMES table, 591
 SYSIBMTS.TSCONFIGURATION table, 591
 SYSIBMTS.TSDEFAULTS table, 591
 SYSIBMTS.TSINDEXES table, 591
 SYSIBMTS.TSLOCKS table, 591
 SYSIN cards, unloading large XML documents, 113
 SYSSTAT.INDEXES catalog view, 391
 system sampling, 419
 System z Application Assist Processors (zAAP), 71-72
 System z Integrated Information Processors (zIIP), 71-72

T

table functions, 200, 557
 table partitioning. *See* range partitioning
 table spaces
 characteristics in DB2 for z/OS, 63
 defined, 34
 page size, 36-39
 XML storage, 51-53, 429
 tables. *See also* catalog tables
 online table moves, 40
 reorganizing XML data, 53-54, 68-69
 XML columns, dropping, 40
 in XML sample database in DB2 for Linux, UNIX, and Windows, 710
 in DB2 for z/OS, 710
 tags (XML), 1-4
 mapping to stringIDs, 31-33, 662
 returning element values without, 163-164
 values versus, 19-21
 target namespaces, 438
 for XML Schemas, 476
 TBSCAN operator (execution plans), 402
 technical support, xxvi.
 See also resources for information
 TEMP operator (execution plans), 402
 terminating characters changing, 549
 CLP option for, 706
 test on pureXML, 675-702
 testing stored procedures, 555-556
 text files as input parameters for CLP, 708
 text indexes (DB2 Net Search Extender)
 altering, 580
 creating, 572-579, 591-592
 reorganizing, 579-580
 updating, 579-580
 text nodes, 29-30
 concatenation, 30
 constructing, 290
 index eligibility and, 375-376
 text searches. *See* DB2 Text Search; full-text searches
 text() node test, 134
 time functions in XQuery, 224-226
 time zone indicators, 210
 TIMESTAMP index data type, 369
 tokenize function (XQuery), 217-218
 TQ operator (execution plans), 402
 transform expression (XQuery), 190, 325
 XML attribute values, replacing, 327-328
 XML element values, replacing, 326-327
 XML node values, replacing
 with computed values, 329-331
 multiple values, 328-329
 with parameter markers, 328
 transformation functions, 579
 transforming
 XML documents, 352-358
 with XQuery, 203
 transition variables, 561
 transitivity of value comparisons, 156
 translate function (XQuery), 218
 traversing XML documents, 197

trees of nodes, 28-30
 storage of, 30-33
 triggers, 523-525, 561-564
 delete triggers, 563
 executing, 547
 IMPORT utility and, 573
 insert triggers, 562-563
 LOAD utility and, 573
 update triggers, 564
 troubleshooting
 empty XPath query
 results, 134
 SQL/XML predicates,
 181-182
 truncated XML document
 display, 83
 avoiding, 138
 type constructors, 212
 type errors
 avoiding in XMLTABLE
 function, 168-169
 in XQuery, 208-212

U

UCA (Unicode Collation Algorithm), 252
 UDFs (user-defined functions),
 547, 556-561
 benefits of, 547
 executing, 547
 extracting
 repeating XML element
 values, 557-558
 XML element/attribute
 values, 557
 inserting XML documents
 from files, 79-82
 shredding XML data,
 558-559
 updating XML documents,
 559-561
 in XQuery, 229-230
 unabbreviated syntax in
 XPath, 157
 underscore character (`_`) in
 wildcard searches, 583

Unicode
 explained, 598
 locale-aware collations, 252
 UTF-8, 27, 597-598
 UTF-16, 598
 UTF-32, 598
 Unicode Byte-Order Mark
 (BOM), 599
 Unicode Collation Algorithm
 (UCA), 252
 union keyword, 154
 union of sequences in XPath,
 154-155
 UNION operator (execution
 plans), 402
 union operator (`|`) in XPath, 585
 UNIONA operator (execution
 plans), 414
 UNIQUE operator (execution
 plans), 402
 unique XML indexes, 364-365
 Universal Resource Identifier
 (URI), 438-439
 UNIX. *See* DB2 for Linux,
 UNIX, and Windows
 UNLOAD utility, 67, 112
 unloading XML documents,
 111-114
 unmarshalling, 629
 update cursors, modifying
 XML documents in, 350-351
 UPDATE INDEX command
 (DB2 Net Search Extender),
 579-580
 UPDATE operator (execution
 plans), 414
 UPDATE statement. *See also*
 XQuery Update Facility
 replacing XML documents,
 322-324
 validation, 518-519,
 542-543
 update triggers, 564
 UPDATE XMLSCHEMA
 command, XML Schema
 evolution with, 495-498

updating. *See also* modifying;
 replacing
 text indexes
 automatic updates,
 574-576
 with DB2 Net Search
 Extender, 579-580
 XML data with XML
 namespaces, 463-469
 XML documents, 433
 in DB2 for z/OS,
 351-352
 with UDFs, 559-561
 XML elements/attributes
 with stored procedures,
 554-555
 upper-case function
 (XQuery), 221
 upper-case() function, 155
 “upsert” operations, 342, 560
 URI (Universal Resource
 Identifier), 438-439
 USC-2, 598
 user-defined functions.
 See UDFs
 user-defined XML indexes,
 664-666
 UTF-8, 27, 597-598
 UTF-16, 598
 UTF-32, 598
 utilities
 monitoring performance of,
 427-428
 XML support in DB2 for
 z/OS, 65-67
 CHECK DATA utility,
 69-70
 REORG utility, 68-69
 REPORT TABLE-
 SPACESET utility,
 67-68

V
 -v CLP option, 708
 valid XML documents
 determining XML Schemas
 for, 538-540

- well-formed documents
 - versus, 473
 - validation, 8, 473
 - application-centric versus database-centric, 545
 - checking XML documents for, 534-535
 - in DB2 for z/OS, 540-544
 - DB2 for Linux, UNIX, and Windows versus, 543-544
 - for existing XML documents, 543
 - with INSERT statement, 541-542
 - with UPDATE statement, 542-543
 - dropped XML Schemas and, 493
 - during loading or importing, 116
 - during shredding process, 312
 - enforcing
 - with check constraints, 520-523
 - with triggers, 523-525
 - error handling, 525-529
 - of existing XML documents, 535-538
 - on INSERT, 514-517
 - with LOAD and IMPORT utilities, 530-534
 - against default XML Schemas, 532
 - against multiple XML Schemas, 530-532
 - against single XML Schema, 530-531
 - overriding XML Schema references, 532-534
 - schema location hints, 534
 - performance considerations, 434
 - removing from XML documents, 540
 - retaining invalid XML documents, 519-520
 - space consumption
 - and, 51
 - on UPDATE, 518-519
 - when to use, 474
 - whitespace preservation and, 517
 - XML Schema evolution
 - with/without, 494-495
 - value comparison operators in XPath, 156-157
 - value predicates, 147, 153
 - values
 - attribute values, attribute nodes versus, 136
 - of repeating XML elements, extracting, 557-558
 - updating in XML documents with namespaces, 464-465
 - of XML attributes
 - extracting, 557
 - replacing, 327-328
 - of XML elements
 - extracting, 557
 - replacing, 326-327
 - of XML nodes, replacing
 - with computed values, 329-331
 - multiple values, 328-329
 - with parameter markers, 328
 - values (XML)
 - converting to binary SQL types, 187-188
 - ordering result sets by, 186-187
 - retrieving in relational format, 165-176
 - tags versus, 19-21
 - values (XQuery Data Model), 128
 - VARCHAR HASHED index data type, 368-369, 433
 - VARCHAR(n) index data type, 367-368
 - variables
 - host variables, 613-614
 - in stored procedures, 548
 - viewing XML document structure, 703-705
 - views. *See* catalog views; relational views
 - Visual Explain tool, 396
 - execution plans, obtaining, 400-401, 411-413
 - Visual Studio, IBM Database Add-ins for Visual Studio, 656
 - volatility of schema, 12
- W**
- WebSphere Replication Server, 119
 - well-formed XML documents, 4, 76
 - valid documents versus, 473
 - where clause (FLWOR expressions), 194
 - whitespace
 - in XML documents, 89-94
 - changing default preservation option, 93-94
 - data storage for compliance, 94
 - preserving, 91-93
 - types of, 90
 - preserving
 - during import, 108
 - validation and, 517
 - retaining in CLP, 527
 - stripping, 78
 - wildcard searches, 583
 - wildcards
 - in full-text searches, 594
 - index eligibility and, 376-377

for namespace queries, 449-450
 in XPath, 140-141
 Windows. *See* DB2 for Linux, UNIX, and Windows
 work directories, locating with index directories, 574

X

XANDOR operator (execution plans), 402-403
 XDBDECOMPXML stored procedure, 313-314
 XDB_DECOMP_XML_FROM_QUERY stored procedure, 315-317
 XDS (XML Data Specifiers), 99
 XISCAN operator (execution plans), 403
 XIXAND operator (execution plans), 414
 XIXOR operator (execution plans), 414
 XIXSCAN operator (execution plans), 414
 XML (eXtensible Markup Language), 1
 application development. *See* application development
 applications, best practices, 434-435
 attributes. *See* attributes (XML)
 CLP options
 list of, 706
 usage examples, 706-707
 documents. *See* documents (XML)
 for data exchange, 1
 for data storage, 2
 pureXML versus alternative storage methods, 10-11

indexes. *See* indexes (XML)
 monitoring performance, 424
 of database utilities, 427-428
 with snapshot monitor, 424-427
 namespaces. *See* namespaces (XML)
 performance. *See* performance
 pureXML. *See* pureXML
 relational data versus, 4-7
 when to use XML data, 11-13
 reorganizing table data, 53-54, 68-69
 schemas, best practices, 434
 as self-describing data format, 19
 as standard, xxiii, xxv, 1
 tags. *See* tags (XML)
 values
 converting to binary SQL types, 187-188
 ordering result sets by, 186-187
 retrieving in relational format, 165-176
 tags versus, 19-21
 XML 1.0 standard, 2
 XML 1.1 standard, 2
 XML aggregation. *See* aggregation
 XML column references. *See* column references (XML)
 XML columns. *See* columns (XML)
 XML compression. *See* compression
 XML construction
 with attribute expressions, 206
 with computed values, 202-204
 with conditional expressions, 205

direct XML
 construction, 202
 with multiple nesting levels, 206-207
 with predicates, 204-205
 with XML aggregation, 207-208
 with XML namespaces, 460-463
 XML data
 converting relational data to, 267
 inserting in XML columns, 294-295
 with SQL/XML publishing functions, 268-290
 XML declarations for, 292-294
 with XQuery constructors, 290-292
 generating rows/columns from, 165-166
 querying. *See* querying XML data
 statistics collection
 in DB2 for Linux, UNIX, and Windows, 418-419
 in DB2 for z/OS, 417-418
 with db2cat utility, 419-423
 truncation, avoiding, 138
 XML data binding
 to Java objects, 629
 pureQuery and, 631
 XML Data Specifiers (XDS), 99
 XML data type, 7-9, 160
 XML declarations. *See* declarations (XML)
 XML document trees, 28-30
 storage of, 30-33
 XML encoding. *See* encoding (XML); Unicode

- XML joins, relational joins
 - versus, 7, 241
- XML manipulation
 - in .NET applications, 633-635
 - in stored procedures, 548-556
 - hybrid XML data storage, 550-553
 - loops and cursors, 553-554
 - testing, 555-556
 - updating XML elements/attributes, 554-555
 - with triggers, 561-564
 - delete triggers, 563
 - insert triggers, 562-563
 - update triggers, 564
 - in UDFs, 556-561
 - extracting repeating XML element values, 557-558
 - extracting XML element/attribute values, 557
 - shredding XML data, 558-559
 - updating XML documents, 559-561
- XML predicates. *See* predicates
- XML publishing functions. *See* publishing functions (SQL/XML)
- XML sample database
 - creating, 709-710
 - customer table contents, 710-712
 - product table contents, 712-713
 - purchaseorder table contents, 713-714
- XML Schema, xxiii, 2, 471
 - annotated schema shredding, 306-318
 - advantages/disadvantages of, 301
 - annotating XML Schema, 306-310
 - defining annotations in Data Studio Developer, 311
 - registering annotated schemas, 311-312
 - shredding multiple XML documents, 315-318
 - shredding single XML documents, 312-315
 - custom versus industry standard, 474-476
 - DB2 for z/OS versus DB2 for Linux, UNIX, and Windows, 510-511
 - determining for validated XML documents, 538-540
 - DTDs versus, 501
 - editing in Data Studio Developer, 653
 - exporting
 - information with db2look utility, 122
 - XML documents containing, 105-106
 - flexibility of, 5-6
 - granting/revoking usage privileges, 499-500
 - identifiers, 516
 - with multiple schema documents, 479-482
 - in .NET applications, handling, 636
 - as optional in DB2, 8
 - parts of, 476-478
 - reasons for using, 472-473
 - referencing, 484
 - registering, 483-491
 - in CLP (command-line processor), 484-486
 - error handling for, 490-491
 - identifiers, 483
 - with JDBC, 488
 - with shared schema documents, 489-490
 - steps in, 483
 - with stored procedures, 486-487
 - removing from XSR, 492-493
 - target namespaces, 438
 - valid versus well-formed XML documents, 473
 - validation. *See* validation when to validate, 474
- XML Schema evolution, 493-498
 - with document validation, 494-495
 - with UPDATE XMLSCHEMA command, 495-498
 - without document validation, 494
- XML Schema Repository (XSR), 483, 502-503, 667, 672. *See also* registering XML Schemas
 - catalog tables/views, 503-508
 - queries against, 508-510
 - registering annotated schemas, 311-312
- XML storage, 429-430
 - for compliance 94
 - in DB2 for Linux, UNIX, and Windows, 33-41
 - dropping XML columns, 40
 - in DB2 9.7 release, 40-41
 - storage objects, types of, 33-35
 - table space page size, 36-39
 - in DB2 for z/OS, 60-73
 - CHECK DATA utility, 69-70
 - limiting memory consumption, 71

- multiple XML
 - columns, 64
 - naming conventions, 64-65
 - offloading XML parsing, 72-73
 - REORG utility, 68-69
 - REPORT TABLE-SPACESET utility, 67-68
 - storage objects, types of, 61-62
 - table space characteristics, 63
 - utilities for, 65-67
- inlining, 41-48
 - benefits of, 47-48
 - drawbacks of, 48
 - monitoring and configuring, 43-47
- MDC (multidimensional clustering), 58-59
- partitioned databases, 59-60
- range partitioning, 57-58
- space consumption of, 51-53
- space management
 - example, 54-57
- XML System Services (XMLSS), 72
- XML to HTML transformation, 356-358
- XML-related catalog tables, 667-673
 - for XML indexes, 671-672
 - XML Schema Repository (XSR), 672
 - for XML storage objects, 667-670
- XML-related catalog views, 661-667
 - SYSCAT.COLUMNS, 661-662
 - SYSCAT.INDEXES, 663-664
 - SYSCAT.INDEXXML-PATTERNS, 664-666
 - SYSIBM.SYSXMLPATHS, 663
 - SYSIBM.SYSXML-STRINGS, 662-663
 - XML Schema Repository (XSR), 667
- XML-to-relational joins, 239, 248-250
- XML-to-XML joins, 239
 - outer joins, 250-252
 - in SQL/XML, 242-247
 - in XQuery, 240-242
- XML2CLOB function (SQL/XML), 290
- xml:space attribute, 91-92
- XMLAGG function (SQL/XML), 160, 207, 277-283
 - XMLCONCAT, XMLFOREST compared, 284
- XMLATTRIBUTES function (SQL/XML), 160, 275-277
- XMLCAST function, 119, 160, 163, 186-187
 - code page conversion example, 604
- XMLCOMMENT function (SQL/XML), 290
- XMLCONCAT function (SQL/XML), 270
 - XMLAGG, XMLFOREST compared, 284
- XmlDocument class (.NET), 634
- XMLDOCUMENT function, 117, 119, 294-295
- XMLELEMENT function (SQL/XML), 160, 268-273, 460-462
- XMLEXISTS predicate, 160, 177-182, 188, 431
- XMLFOREST function (SQL/XML), 272-273
 - XMLAGG, XMLCONCAT compared, 284
- XMLGROUP function (SQL/XML), 286-289
- XMLNAMESPACES function, 453, 460-462
- XMLPARSE function, 92-93, 119, 160, 354
- XMLPATTERN function in index definitions, 363
- XMLPI function (SQL/XML), 290
- XMLQUERY function, 119, 160-165, 188, 430
 - filtering conditions, 587
 - index eligibility and, 385
 - returning
 - element values without XML tags, 163-164
 - repeating elements, 164-165
 - XML column references in, 162-163
- XMLReader class (.NET), 634
- XMLROW function (SQL/XML), 286-289
- XMLSERIALIZE function, 83, 86, 119, 160, 293, 435, 640
- XMLSpy, 657
- XMLSS (XML System Services), 72
- XMLTABLE function, 160, 165-176, 188
 - advantages/disadvantages of, 300
 - aggregation and grouping queries, 234-236
 - code page conversion example, 604
 - generating rows/columns from XML data, 165-166
 - namespace declarations, 452-453
 - numbering rows based on repeating elements, 173-174
 - optional elements, handling, 167-168

- pureQuery and, 631
- returning
 - multiple repeating elements, 174-176
 - repeating elements, 169-173
- shredding XML documents with, 301-306
- splitting XML documents, 116-118
- type errors, avoiding, 168-169
- XMLTEXT function (SQL/XML), 290
- XMLVALIDATE function, 119, 160, 514-519, 535-536
- XMLXSROBJECTID function, 492, 535, 538-539
- XPath, xxiii, 8, 126. *See also* XQuery
 - axes, 157
 - comparing with FLWOR expressions and SQL/XML, 196-202
 - comparison operators, 156-157
 - construction of sequences, 154-155
 - data() function, 134-135
 - dot notation, 151-153
 - double slash (//), 141-142
 - dynamic expressions, 185-186
 - embedding in SQL statements, 127
 - empty results, reasons for, 134
 - executing in DB2, 137-140
 - existential semantics, 147-148
 - file system navigation analogy, 133
 - full-text searches in, 582
 - functions, 155
 - logical expressions, 148-151
 - node tests, 133
 - positional predicates, 153-154
 - predicates, 142-146
 - usage with SQL/XML, 177-182
 - sample data for examples, 131-132
 - simple query examples, 133-136
 - slash (/), 141
 - SQL/XML versus, 201
 - string() function, 135
 - text() node test, 134
 - unabbreviated syntax, 157
 - union of sequences, 154-155
 - union operator (|), 585
 - wildcards, 140-141
- XPath expressions
 - best practices, 430
 - full-text searches with, 593
- XPath queries, design decisions and, 17-18
- XQuery, xxiii, 8, 126. *See also* XPath
 - arithmetic expressions, 212-214
 - attribute expressions in XML construction, 206
 - “between” predicates, 431
 - computed value XML construction, 202-204
 - conditional expressions in XML construction, 205
 - constructors, 290-292
 - XML namespaces and, 462-463
 - contains function, 587
 - data types, cast expressions, type errors, 208-212
 - direct XML construction, 202
 - with embedded SQL, 127
 - embedding
 - in SQL statements, 127
 - SQL in, 227-228
 - FLWOR expressions, 191-196
 - comparing with XPath and SQL/XML, 196-202
 - join queries in, 247
 - full-text searches, 582, 592
 - functions, 214-226
 - Boolean functions, 226
 - date and time functions, 224-226
 - namespace and node functions, 222-224
 - numeric and aggregation functions, 218-220
 - sequence functions, 220-222
 - string functions, 215-218
 - grouping queries in, SQL/XML versus, 237-239
 - join queries, XML-to-XML joins, 240-242
 - let and return clauses, index eligibility and, 386-387
 - modifying XML documents in, 346-349
 - multiple nesting levels in XML construction, 206-207
 - namespace and node functions, 445
 - namespace declarations, 448-450
 - nesting with SQL, 257-258
 - outer joins, 250-252
 - overview, 190
 - predicates in XML construction, 204-205
 - sample data for examples, 131-132
 - SQL functions and UDFs in, 229-230
 - SQL/XML versus, 201

- as stand-alone
 - language, 127
- in stored procedures, 554
- XML aggregation in XML
 - construction, 207-208
 - XSLT versus, 353
- XQuery 1.0 and XPath 2.0
 - Data Model, 126, 128-131
 - sequences
 - constructing, 128-130
 - as input/output, 130-131
- xquery keyword, 137
- XQuery Update Facility, 9, 324-326
 - XML attribute values, replacing, 327-328
 - XML element values, replacing, 326-327
 - XML elements/attributes, renaming, 334-335
 - XML node values, replacing
 - with computed values, 329-331
 - multiple values, 328-329
 - with parameter markers, 328

- XML nodes
 - deleting, 333-334
 - inserting, 335-340
 - modifying multiple, 343-346
 - repeating/missing, 340-343
 - replacing, 331-332
- XSCAN operator (execution plans), 402
- XSL (eXtensible Stylesheet Language), 352
- XSLT (eXtensible Stylesheet Language Transformation), 352-358
 - XML to HTML transformation, 356-358
 - XQuery versus, 353
 - XSLTRANSFORM function, 353-356
- XSLTRANSFORM function, 352-356
- XSR (XML Schema Repository), 483, 502-503, 667, 672. *See also* registering XML Schemas catalog tables/views, 503-508

- queries against, 508-510
- registering DTDs, 501
- removing XML Schemas from, 492-493
- XSR Objects, 483
- XSR_GET_PARSING_DIAGNOSTICS stored procedure, 525-528

Y-Z

- z/OS. *See* DB2 for z/OS
- zAAP (System z Application Assist Processors), 71-72
- zeros, leading zeros in XML element construction, 285-286
- zIIP (System z Integrated Information Processors), 71-72