Information Management

# Extremely pureXML in DB2 10 for z/OS

Develop Java and COBOL applications accessing XML and SQL data

Administer your XML and SQL data

Choose the best options for installation and use

Paolo Bruni
Neale Armstrong
Ravi Kumar
Kirsten Ann Larsen
Tink Tysor
Hao Zhang

ibm.com/redbooks

Redbooks

International Technical Support Organization

# Extremely pureXML in DB2 10 for z/OS

January 2011

SG24-7915-00

**Note:** Before using this information and the product it supports, read the information in "Notices" on page xvii.

**First Edition (January 2011)**

This edition applies to Version 10.1 of IBM DB2 for z/OS (program number 5605-DB2).

**Note:** This book is based on a pre-GA version of a program and may not apply when the program becomes generally available. Consult the program documentation or follow-on versions of this publication for more current information.

**© Copyright International Business Machines Corporation 2011. All rights reserved.**
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

© Copyright IBM Corp. 2011. All rights reserved.

# Figures

© Copyright IBM Corp. 2011. All rights reserved.

# Tables

© Copyright IBM Corp. 2011. All rights reserved.

# Examples

© Copyright IBM Corp. 2011. All rights reserved.

# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
*IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright IBM Corp. 2011. All rights reserved.

# Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
CICS®
DataPower®
DB2 Connect™
DB2®
Domino®
DRDA®
ESCON®
FICON®
IBM®
IMS™
Informix®
InfoSphere™
iSeries®
MVS™
Optim™
OS/390®
OS/400®
PR/SM™
pureXML®
RACF®
Rational®
Redbooks®
Redpaper™
Redbooks (logo) ®
S/390®
System z®
VisualAge®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows NT, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

# Preface

The DB2® pureXML® feature offers sophisticated capabilities to store, process and manage XML data in its native hierarchical format. By integrating XML data intact into a relational database structure, users can take full advantage of DB2's relational data management features.

In this IBM® Redbooks® publication, we document the steps for the implementation of a simple but meaningful XML application scenario. We have chosen to provide samples in COBOL and Java™ language. The purpose is to provide an easy path to follow to integrate the XML data type for the traditional DB2 for z/OS® user.

We also add considerations for the data administrator and suggest best practices for ease of use and better performance.

## The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

**Paolo Bruni** is a DB2 Information Management Project Leader at the International Technical Support Organization based in the Silicon Valley Lab. He has authored several IBM Redbooks publications about DB2 for z/OS and related tools and has conducted workshops and seminars worldwide. During Paolo's many years with IBM, in development and in the field, his work has been mostly related to database systems.

**Neale Armstrong** is a Consultant IT Specialist at IBM in the United Kingdom, responsible for technical support for System z® Information Management products. He has 24 years of experience in DB2 solutions on z/OS and distributed platforms. He holds a degree in Physics from the University of Bristol. His areas of expertise include database federation, replication and event publishing, for DB2, IMS™, and VSAM data sources, which can more generally be referred to as "database plumbing." He has co-authored three previous IBM Redbooks publications.

**Ravi Kumar** is a Senior Instructor and Specialist for DB2 with IBM Software Group, Australia. He has approximately 25 years of experience in DB2. He was on assignment at the International Technical Support Organization, San Jose Center, as a Data Management Specialist from 1994 to 1997. He has co-authored many IBM Redbooks publications including *DB2 UDB for z/OS Version 8 Everything You Ever Wanted to Know, ... and More*, SG24-6079, *DB2 9 for z/OS Technical Overview,* SG24-7330, and *DB2 10 for z/OS Technical Overview,* SG24-7892. He is currently on virtual assignment as a Course Developer with the Education Planning and Development team, Information Management, IBM Software Group, U.S.A.

**Kirsten Ann Larsen** is a Senior IT Specialist and Technical Lead with IT Delivery at IBM in Denmark. She has 14 years of experience with DB2 for z/OS and has co-authored the IBM Redbooks publication *Securing DB2 and Implementing MLS on z/OS*, SG24-6480. She holds a Master's degree in Computer Science from Aarhus University. She has worked with XML since pureXML support was included with the release of DB2 9 in 2007 and has co-authored a number of articles about XML.

© Copyright IBM Corp. 2011. All rights reserved.

**Tink Tysor** is President of Bayard Lee Tysor, Inc. in the U.S.A. He has 40 years of experience in programming, the last 14 years specializing in DB2. He holds a degree in Economics from American University. His areas of expertise include SQL, DB2 Database Administrating, and Data Modeling. He has written and presented extensively on SQL for DB2 including the IBM Redbooks publication *DB2 for z/OS Tools for Database Administration and Change Management,* SG24-6480.

**Hao Zhang** is a Software Engineer in the IBM China Software Development Lab. He has over six years of experience in DB2 QA field. He participated in testing several DB2 pureXML features in DB2 9 and DB2 10 for z/OS, and presented DB2 9 pureXML support to the Chinese DB2 Users' Group (CDUG) in 2009. His areas of expertise include distributed area in JCC driver, temporal table, and XML.

Thanks to the following people for their contributions to this project:

Rich Conway
Bob Haimowitz
Emma Jacobs
Diane Sherman
International Technical Support Organization

Mengchu Cai
Li Chen
Jason Cu
Thanh Dao
Eric Katayama
Andrew Lai
Susan Malaika
Gary Mazo
Roger Miller
Jinfeng Ni
Matthias Nicola
Bryan Patterson
Tom Ross
Guogen Zhang
IBM Silicon Valley Lab

Heidi Arnold
IBM Boeblingen

Michael Schwartzbach
Aarhus University

Rick Butler
BMO Toronto

Lee Ackerman
IBM Ottawa

Nagesh Subrahmanyam
IBM India

# Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

**ibm.com**/redbooks/residencies.html

# Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  **ibm.com**/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

# Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

# Introduction

This chapter provides an introduction to XML technology, its importance in the IT business, and the contents of the book.

This chapter contains the following topics:

- Importance of XML data
- XML introduction
- What is in this book

© Copyright IBM Corp. 2011. All rights reserved.

## 1.1 Importance of XML data

XML technology has become pervasive in virtually all industries and sectors, owing to its versatility and neutrality for exchanging data among diverse devices, applications, and systems from various vendors. These qualities of XML, along with its easy-to-understand self-describing nature, ability to handle structured, semi-structured, and unstructured data, and support for Unicode, have made XML a universal standard for data interchange.

### 1.1.1 Growth of XML

Nearly every company today is exposed to XML in some form. The amount of XML data that organizations have to handle is growing at a rapid rate. In fact, the volume of XML data is growing faster than the traditional data that typically resides in relational databases. The following factors are fueling the growth of XML data:

- XML-based industry and data standards
- Service-oriented architectures (SOA) and web services
- Web 2.0 technologies such as XML feeds and syndication services

#### XML-based industry standards

Almost every industry has multiple standards based on XML, and numerous cross-industry XML standards exist also. Examples of XML-based industry standards are as follows:

- ACORD: XML for the Insurance Industry

  http://www.acord.org/

- FPML: Financial products

  http://www.fpml.org/

- HL7: Healthcare

  http://www.hl7.org/

- IFX: Interactive Financial Exchange

  http://www.ifxforum.org/

- IXRetail: Standard for retail operation

  http://www.nrf-arts.org/

- XBRL: Business reporting and accounting

  http://www.xbrl.org/

- NewsML: News and publication

  http://www.newsml.org/

These standards facilitate purposes such as the exchange of information between the various players within these industries and their value-chain members, data definitions for ongoing operations, and document specifications. More companies are adopting such XML standards or are being compelled to adopt them to be able to stay competitive, improve efficiencies, communicate with their trading partners or suppliers, or simply to perform daily tasks.

#### SOA and web services

Service-oriented frameworks and deployments are growing in popularity, owing to their ability to integrate systems, permit reuse of resources, and respond quickly to changing market conditions, allowing companies to save money and improve competitiveness. In service-oriented architectures, consumers and service providers exchange information using messages. These messages are invariably encapsulated as XML. As such, XML can provide the “plumbing” in SOA environments as illustrated in Figure 1-1. Therefore, the drive towards information as a service and rapid adoption of SOA environments is also stimulating the growth of XML.

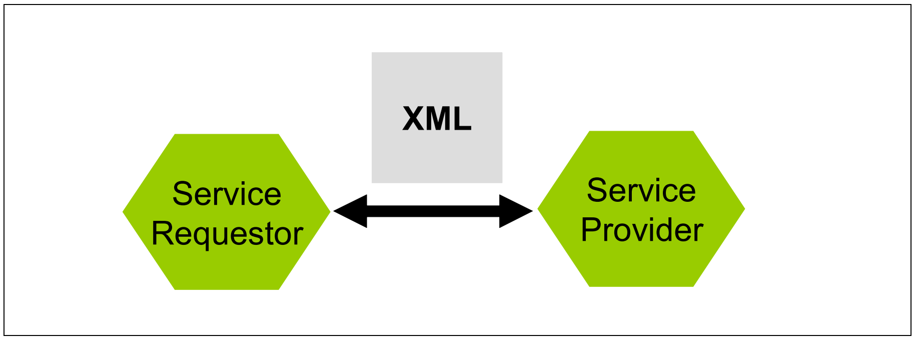


*Figure 1-1 XML: The foundation for web services*

### Web 2.0 technologies

Syndication is considered to be the heartbeat of Web 2.0, the next generation of the Internet. Atom and RSS feeds can be found abundantly on the web, allowing the user to subscribe to them and be kept up-to-date about all kinds of web content changes, such as news stories, articles, wikis, and audio and video files.

Content for these feeds is rendered as XML files and can contain links, summaries, full articles, and even attached multimedia files such as podcasts. Syndication and web feeds are transforming the web as we know it. New business models are emerging around these technologies. As a consequence, XML data now exists not only in companies that adopt XML industry standards, or enterprises that implement SOAs, but also on virtually every web-connected desktop.

## 1.1.2 The value of XML data

As a result of XML industry standards becoming more prevalent, the drive towards SOA environments, and rapid adoption of syndication technologies, more XML data is being generated every day as web feeds, purchase orders, transaction records, messages in SOA environments, financial trades, insurance applications, and other industry-specific and cross-industry data. That is, XML data and documents are becoming an important business asset containing valuable information (such as customer details, transaction data, order records, and operational documents).

The growth and pervasiveness of XML assets presents challenges and opportunities for companies. When XML data is harnessed, and the value of the information it contains is unlocked, it can translate into opportunities for organizations to streamline operations, derive insight, and become agile.

However, as XML data becomes more critical to the operations of an enterprise, it presents challenges in that XML data must be secured, maintained, searched, and shared. Depending on its use, XML data might also have to be updated, audited, and integrated with traditional data. All these tasks must be done with the reliability, availability, and scalability afforded to traditional data assets.

That is, to unleash the potential of XML data requires storage and management services similar to what enterprise-class relational database management systems such as DB2 have been providing for relational data.

## 1.2 XML introduction

This brief introduction to Extensible Markup Language (XML) is extracted from *XML on z/OS and OS/390: Introduction to a Service-Oriented Architecture*, SG24-6826.

The idea of universal data formats is not new. Programmers have been trying to find ways to exchange information between various computer programs for a long time. Standard Generalized Markup Language (SGML) was developed to achieve this. *SGML* can be used to *mark up data*, that is, to add metadata in a way that allows data to be *self-describing*. SGML is meta-language.

The markup process involves using *tags* to identify pieces of information in a document. Tags are names (strings of characters) surrounded by angle brackets (< and >). Every piece of data that is encoded has a start tag and an end tag, for example, `<town> patiya</town>`. The start and end tags help software to process the encoded information, because it clearly delineates where certain pieces of information start and where they end.

SGML does not prescribe any particular markup; instead, it defines how any markup language can be formally specified.

The most popular SGML application is Hypertext Markup Language (HTML), the markup language that rules the web. The HTML specification is owned by World Wide Web Consortium (W3C). However, various browser vendors introduced a number of incompatible tags to HTML, which are outside the scope of the original HTML specifications. These tags create problems for developers when they author web pages because the developers must consider what browser will display the pages. Although HTML has been very successful for displaying information on browsers, it was not found to be useful in describing the data that it represents, meaning it did not have the metadata capability that is essential for a self-describing data document.

Furthermore, SGML is quite inefficient and cumbersome when it is used to encode complex data structure. As a result, a need arose to develop a more lightweight markup language, so W3C developed the specification for XML. XML is similar to SGML in that it preserves the notion of *general markup*. There are very few optional features, and most SGML features that were deemed difficult to implement have been dropped.

In this section, we examine the following topics:

- XML definitions
- Document validity and well-formedness
- XML Schema
- Extensible Stylesheet Language
- XPath
- XQuery
- XHTML
- XSL, XSLT, Xpath, and XHTML examples

### 1.2.1 XML definitions

XML is a system-independent standard for the representation of data. XML is not simply some new version of HTML; it is different from HTML. As with HTML, XML has tags, and in these tags it encloses data. Where an HTML tag says something similar to “display this data in bold font” (`<b>...</b>`), an XML tag acts like a field name in your program. It puts a label on a piece of data that identifies it (for example `<message>...</message>`). This difference is the first of a number of differences between the languages.

In XML, you can create the tags you want, with only a small number of restrictions, and these tags are used by a program (parser) to process the data enclosed between them.

Example 1-1 shows a simple XML document.

*Example 1-1 An XML document*

```
<?xml version=”1.0”?>
<!DOCTYPE JavaXML:EmployeeList SYSTEM “DTD\JavaXML.dtd”>
<JavaXML:employeeList xmlns:JavaXML=”http://www.ibm.com”>
   <JavaXML:Employee action=”add”>
      <JavaXML:firstName>David</JavaXML:firstName>
      <JavaXML:secondName>Sanchez Carmona</JavaXML:secondName>
      <JavaXML:age>20</JavaXML:age>
   </JavaXML:Employee>
   <JavaXML:Employee action=”delete”>
      <JavaXML:firstName>Jose Luis</JavaXML:firstName>
      <JavaXML:secondName>Fernandez Lastra</JavaXML:secondName>
   </JavaXML:Employee>
</JavaXML:employeeList>
```

A client, for example, could use a web browser to fill out a form, entering the names of the employees to add or delete. The data could then be sent to a web application that could process the XML document and extract the data, generating the necessary updates, for example, on a DB2 table.

As this example illustrates, the rules are very few: each tag must have an enclosing tag, and not much more. The tags are invented tags, which means that they are free-form.

Text is system-independent, and because XML is very flexible and is based only on text, it is used as the main way to transport data between various environments.

Often, XML documents are automatically generated by tools, and in many situations we need these XML documents to follow rules we create. We use other documents, containing XML data definitions in which we specify our restrictions, to accomplish this.

Document type definition (DTD) is a set of markup declarations that define a document type for SGML-family markup languages (SGML, XML, HTML). DTD is described in “Document type definition” on page 7.

*XML Schema*[1] is another rules language that aims to provide more complex semantic rules. It also introduces new semantic capabilities, such as support for namespaces and type-checking within an XML document. XML Schema is described in 1.2.3, “XML Schema” on page 9.

## 1.2.2 Document validity and well-formedness

XML documents can be *well-formed*, or they can be *well-formed and valid.* These are two very important rules that do not exist for HTML documents. These strong rules contrast with the more free-style nature of many concepts in XML. The rules are defined briefly as follows:

- A *well-formed* document satisfies a list of syntax rules that are provided in the specification for XML documents.
- A *valid* document contains a reference to a DTD or XML Schema Definition (XSD), and its elements and attributes follow the grammatical rules that the DTD or XSD specifies.

[1] The W3C-recommended schema language for XML is XML Schema; see http://www.w3.org/XML/Schema

Schema languages typically constrain the set of elements that may be used in a document, which attributes may be applied to them, the order in which they may appear, and the allowable parent/child relationships

*XSD schema*, often referred to as XML Schema, is a newer schema language, successor to DTD language. XSD documents are far more powerful than DTDs in describing XML languages. They use a rich data typing system and allow for more detailed constraints on an XML document's logical structure. XSDs also use an XML-based format, which enables the possibility of using ordinary XML tools to help process them. This approach has become the more popular one to working with XSD.

With few exceptions, every DTD can be converted to an equivalent XML Schema.

### Difference between well-formedness and validity

All the constraints are defined in the XML 1.0 recommendation. For more information, see the following website:

http://www.w3.org/XML

Determining whether a particular document is in compliance with these rules is a two-step process. Well-formedness ensures that XML parsers are able to read the document; validity (which implies well-formedness) determines whether an XML document adheres to a DTD or XML Schema. An XML application checks for and rejects documents that are not well-formed before checking whether they comply with validity constraints (VCs).

A document might be well-formed but still not be valid. The following examples illustrate the difference between well-formedness and validity:

- Documents that adhere to rules described in the associated DTD or XSD are valid.
- Documents that carry out the syntactical rules for XML documents are well-formed. These rules have to do with attribute names, which must be unique within an element, and attribute values, which must not contain the character open angle bracket (<), and so on.

Example 1-2 shows a DTD.

*Example 1-2 DTD*

```
<!ELEMENT BANCO (TARJETA+)>
<!ELEMENT TARJETA (Nom, Cod_Cuenta)>
<!ELEMENT Nom (#PCDATA)>
<!ELEMENT Cod_Cuenta ((#PCDATA)>
```

Example 1-3 shows a sample XML document.

*Example 1-3 XML document*

```
<BANCO>
   <TARJETA>
      <NOM>Silvia</NOM>
      <Cod_Cuenta>2562789452</Cod_Cuenta>
   </TARJETA>
</BANCO>
```

The document shown in Example 1-3 is well-formed, but it is not valid according to the sample DTD shown in Example 1-2 because the <NOM> tag is not defined in the associated DTD (tags are case-sensitive).

## Document type definition

A document type definition (DTD) specifies the kinds of tags that can be included in your XML document, the valid arrangements of those tags, and the structure of the XML document. The DTD defines the type of elements, attributes, and entities allowed in the documents, and may also specify limitations to their arrangement. You may use the DTD to make sure you do not create an invalid XML structure because the DTD defines how elements relate to one another within the document's tree structure. You may also use it to define which attributes can be used to define an element and which ones are not allowed.

The DTD can either be stored in a separate file or be embedded within the same XML file. If it is stored in a separate file it may be shared with other documents.

An XML document is not required to have a DTD. DTDs provide parsers with clear instructions on what to check for when they are determining the validity of an XML document. DTDs or other mechanisms, like XML schemas, contribute to the goal of ensuring that the application can easily determine whether the XML document adheres to a given set of rules, beyond the well-formedness rules defined in the XML standard.

DTDs do have limitations, as the following examples describe:

- A DTD makes validating the structure of relatively simple XML documents possible, but that is its limit. A DTD cannot restrict the content of elements, and it cannot specify complex relationships.
- In a DTD, you may specify the structure of the `<heading>` element only one time. There is no context-sensitivity.
- A DTD specification is not hierarchical. For a mailing address that contains several parsed character data (PCDATA) elements, for instance, the DTD might look similar to that shown in Example 1-4. As you can see, the specifications are linear. That fact forces you to devise new names for similar elements in separate settings.

*Example 1-4 Introducing need for namespaces*

```
<!ELEMENT mailAddress (name, address, zipcode)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT zipcode (#PCDATA)>
```

- Because of the non-hierarchical nature of DTD specifications, a lack of clarity exists about what the comments are meant to explain.
- A DTD uses syntax that is differs substantially from XML, so it cannot be processed with a standard XML parser. That means you cannot read a DTD into an XML Document Object Model (DOM)[2]. For example, you cannot modify it, and then write it back out again.

## Namespaces

Before talking about XML Schema, we must first clarify the concept of namespaces. *Namespaces* are used when there is a need to have separate elements with different attributes but with the same name. Depending on the context, a tag is related to an element or to another tag. Example 1-5 on page 8 illustrates this situation.

[2] DOM is a programming interface for HTML and XML documents; it defines the way a document can be accessed and manipulated.

*Example 1-5 The need for namespaces*

```
<widget type="gadget">
   <head size="medium"/>
      <info>
         <head>
          <title>Description of gadget</title>
         </head>
         <body>
          <h1>Gadget</h1>
         </body>
      </info>
</widget>
```

Obviously, there is a problem with the meaning of `<head>` tag. It depends on the context. This situation complicates matters for processors and might even cause ambiguities. We need a mechanism to distinguish between the two `<head>` tags, and apply the correct semantic description to the correct tag. The root of the problem is one common element or attribute name.

A simple solution to the problem is the use of namespaces. Namespaces are a simple and straightforward way to distinguish names that are used in XML documents. If you can specify the related DTD when an element is being validated, the problem is solved.

As you can see in Example 1-6, the `<title>` tag is used twice, but in a different context: once within the `<author>` element and once within the `<book>` element. Note the use of the `xmlns` keyword in the namespace declaration, one for `authr`, one for `bk`. What is interesting is that the XML recommendation does not specify whether a namespace declaration should point to a valid Uniform Resource Identifier (URI), only that it should be unique and persistent.

*Example 1-6 Namespaces*

```
<?xml version="1.0" ?>
<library-entry xmlns:authr="http://sc58ts.itso.ibm.com/Jose/2002/author.dtd"
xmlns:bk="books.dtd">
  <bk:book>
    <bk:title>XML Sample</bk:title>
    <bk:pages>210</bk:pages>
    <bk:isbn>1-868640-34-2</bk:isbn>
    <authr:author>
      <authr:firstname>JuanJose</authr:firstname>
      <authr:lastname>Hernandez</authr:lastname>
      <authr:title>Mr</authr:title>
    </authr:author>
  </bk:book>
</library-entry>
```

In Example 1-6, to illustrate the relationship of each element to a given namespace, we specify the relevant namespace prefix before each element. Prefixes are bound to namespace URIs by attaching the `xmlns:`*`prefix`* attribute to the prefixed element or one of its ancestors. Bindings have scope within the element where they are declared.

After a prefix is applied to an element name, it applies to all descendants of that element unless it is overridden by another prefix. The extent to which a namespace prefix applies to elements in a document is defined as the *namespace scope*.

Example 1-7 is equivalent to Example 1-6 on page 8, but only the necessary namespace prefixes have been used.

*Example 1-7 Namespaces without prefixes*

```
<?xml version="1.0" ?>
<library-entry xmlns:authr="http://sc58ts.itso.ibm.com/Jose/2002/author.dtd"
xmlns:bk="books.dtd">
  <bk:book>
    <title>XML Sample</title>
    <pages>210</pages>
    <isbn>1-868640-34-2</isbn>
    <authr:author>
      <firstname>JuanJose</firstname>
      <lastname>Hernandez</lastname>
      <title>Mr</title>
    </authr:author>
  </bk:book>
</library-entry>
```

Information about namespaces is at the following website:

http://www.w3.org/TR/REC-xml-names

## 1.2.3 XML Schema

The W3C XML Schema Definition (XSD) language is an XML language for describing and constraining the content of XML documents. A *schema* is similar to a DTD in that it defines which elements an XML document can contain, how they are organized, and which attributes and attribute types elements can be assigned. Therefore it is a method to check the validity of well-formed XML documents. For more information, see the following website:

http://www.w3.org/XML/Schema

### DTD and XML Schema

“Document type definition” on page 7 introduces DTD and identifies several of its limitations. In addition to those limitations, we can add the following limitations:

- No constraints exist on character data. If character data is allowed, any character data is allowed.
- The attribute value models are too simple.
- No support exists for namespaces.
- No support exists for schema evolution, extension, or inheritance of declarations.
- It is difficult to write, maintain, and read large DTDs, and to define families of related schemas.
- Setting defaults for attributes is possible, but not for elements.

Therefore, there is a need for a way to specify more complex semantic rules and provide all those tasks that DTDs cannot do, such as type-checking within an XML document. XML Schema provides such functionality; it also introduces semantic capabilities, such as support for namespaces and type-checking.

The main advantages of XML Schemas over DTDs are as follows:

- Schemas use XML syntax.
- It is possible to specify data types.
- Schemas are extensible.

## XML Schema example

Giving a general outline of the elements of a schema is difficult because of the number of elements that can be used according to the W3C XML Schema Definition Language. The purpose of this language is to provide an inventory of XML markup constructs with which to write schemas. Example 1-8 is a simple document which describes the information about a book.

*Example 1-8 A book description*

```
<?xml version="1.0" encoding="UTF-8"?>
<book isbn="0836217462">
   <title>
   Don Quijote de la Mancha
   </title>
   <author>De Cervantes Saavedra, Miguel</author>
   <character>
      <name>Sancho Panza</name>
      <friend-of>El Quijote</friend-of>
      <since>1547-10-04</since>
      <qualification> escudero </qualification>
   </character>
   <character>
      <name>ElbaBeuno</name>
      <since>1547-08-22</since>
      <qualification>Amor Platonico de Don Quijote</qualification>
   </character>
</book>
```

Because the XML Schema is a language, several choices are available to build a possible schema that covers the XML document. Example 1-9 on page 11 is a possible and simple design.

*Example 1-9 XML Schema*

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="character" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="friend-of" type="xs:string" minOccurs="0"
                maxOccurs="unbounded"/>
              <xs:element name="since" type="xs:date"/>
              <xs:element name="qualification" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example 1-9 clearly is an XML document because it begins with the XML document declaration. The schema element opens our schema that is holding the definition of the target namespace. Then, we define an element named `book`, which is the root element in the XML document. We determine it is a complex type because it has attributes and non-text children. With `sequence`, we begin to declare the children elements of the root element `book`. W3C XML Schema lets us define the type of data, and also the number of possible occurrences of an element. For more information about possible values for these types, see the specification documents from W3C.

## XML Schema options

XML Schema Language offers possibilities and alternatives beyond what is shown in Example 1-9. We could develop another schema based on a flat catalog of all the elements available in the instance document and, for each of them, lists of child elements and attributes. As a result, we would have two choices: defining elements and attributes as they are needed, or creating them first and referencing them. The first option has a real disadvantage: the schema could become very difficult to read and maintain when documents are complex.

W3C XML Schema allows us to define data types and use these types to define our attributes and elements. It also allows the definition of groups of elements and attributes. In addition, there are several ways to arrange relationships between elements.

Documentation for XML Schemas can be defined by the `xs:documentation` element, and processing instructions for applications can be included with the `xs:appinfo` element.

As of August 2009, XSD 1.1 is a Candidate Recommendation with significant new features as defined at the following web addresses:

- http://www.w3.org/TR/xmlschema11-1/
- http://www.w3.org/TR/xmlschema11-2/

### 1.2.4 Extensible Stylesheet Language

In previous sections, we covered XML, its syntax, how XML is used to mark up information according to our own vocabularies, how a program can check the validity of an XML document, and so forth. We described how XML can ensure that an application running on any particular platform receives valid data. This way is how we ensure that a program is able to process this data.

However, because XML describes only document syntax, the program does not know how to format this data without specific instructions about style.

The solution is XSL transformations. The Extensible Stylesheet Language (XSL) specification describes powerful tools to accomplish the required transformation of XML data. XSL consists of the following items:

- The XSL Transformations (XSLT) language for transformation
- Formatting Objects (FO), a vocabulary for describing the layout of documents
- XML Path Language (XPath), which XSLT uses as a separate specification that describes a means of addressing XML documents and defining simple queries.

XSLT offers a powerful means of transforming XML documents into other forms, producing XML, HTML, and other formats. It is capable of sorting, selecting, numbering, and has many other features for transforming XML. It operates by reading a style sheet, which consists of one or more templates, then matching the templates as it visits the nodes of the XML document. The templates can be based on names and patterns.

XSLT is increasingly being used to transform XML data into another form, sometimes different XML (for example, filtering out certain data, SQL statements, plain text, and so on), or any other format. Thus, any XML document may be shown in various formats, such as HTML, PDF, RTF, VRML, Postscript, and so forth.

The question is how to access and display the information contained in an XML file. After all, data is useless unless you can use it. This is where XSLT comes into the picture.

A comparison can be made between the relationship of Cascading Style Sheets (CSS)[3] and HTML and the relationship of XSLT and XML. Indeed, XSLT is usually referred to as the *stylesheet language* of XML; however XML and XSLT are far more sophisticated technologies than HTML and CSS.

XSLT is a high-level declarative language. It is also a transforming and formatting language. It behaves in the following way:

1. The pertinent data is extracted from an XML source document and transformed into a new data structure that reflects the desired output. The XSLT markup is commonly called a *stylesheet*. A parser is used to convert the XML document into a tree structure composed of various types of nodes. The transformation is accomplished with XSLT by using pattern-matching and templates. Patterns are matched against the source tree structure, and templates are used to create a result tree.
2. Next, the new data structure is formatted, for example in HTML or as text, and finally the data is ready for display.

Figure 1-2 on page 13 shows the source tree from the XML document shown in Example 1-10 on page 15.

[3] CSS is a mechanism for adding style (such as fonts, colors, spacing) to web documents; see http://www.w3.org/Style/CSS/

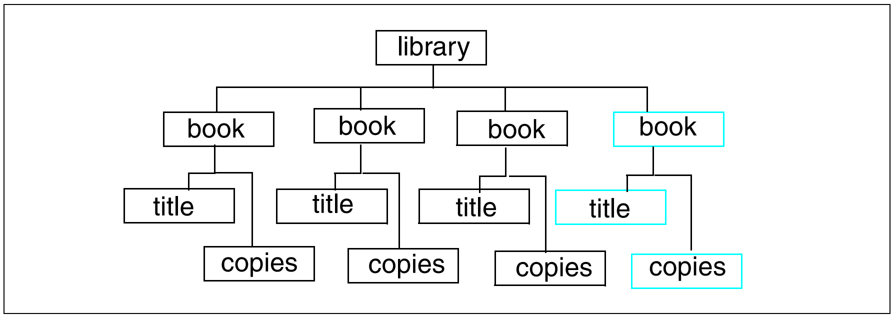


*Figure 1-2 DOM tree*

The result tree after an XSL transformation can be an XHTML document, as shown in Figure 1-3.

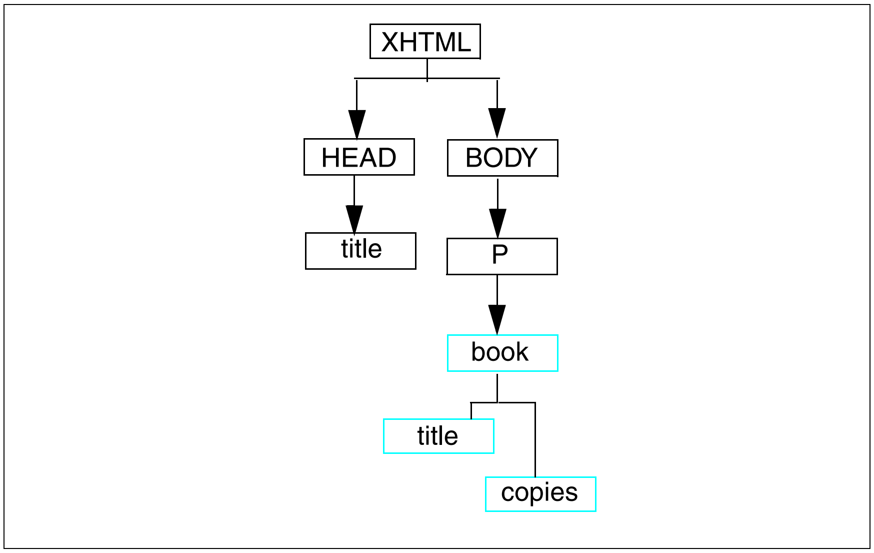


*Figure 1-3 DOM tree after XSL transformation*

Based on how we instruct the XSLT processor to access the source of the data being transformed, the processor will incrementally build the result by adding the filled-in templates. We write our stylesheets, or *transformation specifications*, primarily with declarative constructs, although we can employ procedural techniques if and when needed. We assert the desired behavior of the XSLT processor, based on conditions that are in our source.

**Note:** XSLT manipulates only the source tree; the original XML document is unchanged.

The most important aspect of XSLT is that it allows you to perform extremely complex manipulations on the selected tree nodes by affecting both content and appearance. The final output might bear absolutely no resemblance to the source document. This ability to manipulate the nodes is where XSLT far surpasses CSS.

The W3C set the recommended standards for XSLT Version 1.0. The W3C proposed recommendation for XSL is available at the following address:

`http://www.w3.org/TR/xslt`

### 1.2.5 XPath

XPath is a string syntax for building addresses to the information found in an XML document. We use this language to specify the locations of document structures or data found in an XML document when processing that information using XSLT. XPath allows us from any location to address any other location or content. That is, XPath is a tool used in XSLT to select certain information to be formatted.

XPath 2.0 is the current version of the language. A number of implementations exist but are not as widely used as XPath 1.0. The XPath 2.0 language specification changes several fundamental concepts of the language such as the type system.

For details, see the following address:

http://www.w3.org/TR/xpath20/

XPath expressions are usually built from patterns, which describe a branch of an XML tree. A pattern, therefore, is used to reference one or more hierarchical nodes in a document tree. The XPath patterns that are listed in Table 1-1 are several examples to give you an idea what kinds of items can be selected.

*Table 1-1 XPath patterns*

| Symbol | Meaning |
|---|---|
| / | Root pattern. It refers to an immediate child. |
| // | Separator of steps. It refers to any descendant in the node. By default of axis[a], in the next step, it refers to a child. |
| . | Context item. It refers to the current node. |
| * | Wildcard pattern. It refers to all elements in the actual node. |
| @ | Attribute reference. It refers to an attribute by preceding an attribute name. |
| @* | Attribute reference. It refers to all attributes in the actual node. |

[a] An *axis* defines a node-set relative to the current node.

XPath models an XML document as a tree of nodes, as follows:

- A root node
- Element nodes
- Attribute nodes
- Text nodes
- Namespace nodes
- Processing instruction nodes
- Comment nodes

The basic syntactic construct in XPath is the *expression* (Example 1-10 on page 15). An object is obtained by evaluating an expression, which has one of the following four basic types:

- Node-set (an unordered collection of nodes without duplicates)
- Boolean
- Number
- String

*Example 1-10 XPath*

```
<?xml version="1.0"?>
<!DOCTYPE library system "library.dtd">
<library>
   <book ID="B1.1">
      <title>xml</title>
      <copies>5</copies>
   </book>
   <book ID="B2.1">
      <title>WebSphere</title>
      <copies>10</copies>
   </book>
   <book ID="B3.2">
      <title>great novel</title>
      <copies>10</copies>
   </book>
   <book ID="B5.5">
      <title>good story</title>
      <copies>10</copies>
   </book>
</library>
```

Considering Example 1-10, we can make paths as follows:

- `/book/copies`

  Selects all `copies` element children of `book` elements.

- *`/book//title`*

  Selects all `title` elements in the tree, although `title` elements are not immediate children.

- `/book/@ID`

  Selects all ID attributes for book elements.

However, as we mentioned previously, another possibility is to select elements based on other criteria, such as the following example:

`/library/*/book[title $eq$ "good story"]`

This example selects all `book` elements beyond `library` element, but only if the `title` element matches with `good story`.

## 1.2.6 XQuery

XQuery is a functional language that extends XPath. Its basic building blocks are expressions that are constructed from keywords, operators (symbols), and operands (which are usually other expressions). Expressions can be nested with full generality. An XQuery query is composed of a prologue and a body. The query prologue is optional and consists of declarations that define the execution environment of the query. The query body consists of an expression that provides the result of the query. The input and the output of the query are values (instances) of the XQuery 1.0 and XPath 2.0 Data Model (XDM)[4].

[4] The term *XDM instance* is used, like the term value, to denote an unconstrained sequence of nodes or atomic values in the data model.

Example 1-11 shows a typical XQuery query. An XQuery query is made up of a *prolog* and a *body*. The XQuery prolog is a series of declarations and definitions that together create the required environment for query processing. This XQuery prolog includes a namespace declaration. The XQuery body consists of expressions that specify the intended query result.

*Example 1-11 XQuery query*

```
declare default element namespace "http://sample.name.space.com";
for $cust in db2-fn:xmlcolumn('XPS.DOC')
return
$cust/Name/LastName;
```

## 1.2.7 XHTML

The history of XHTML is very simple: It is derived directly from HTML version 4.01 and is designed to be used with XML. XHTML is part of a whole new suite of “X” technologies, with acronyms such as XML, XPATH, XSL, and XSLT, that are destined to have a profound effect on the Internet.

People often think XML is an extension of HTML, but XHTML is the real extension of HTML.

Several fundamental differences exist between HTML and XHTML that significantly affect how you code with XHTML. HTML is a loose and forgiving language; XHTML demands firm adherence to the rules of grammar.

Fortunately, the syntax and coding rules are very straightforward, easy to implement, and they make sense. The real purpose of these rules is to allow a seamless integration of XHTML with XML and other related X technologies. The rules are summarized as follows:

- All attributes, events, and tags must be written in lower case.
- All elements must be closed.
- The value assigned to an attribute must be enclosed in quotes.
- No attribute may be minimized.
- All elements must be properly nested.
- XHTML documents must be well-formed.
- A DOCTYPE declaration must exist.

This last rule implies that there must be a DTD to validate the XHTML document. HTML has become an XML document.

### XHTML document types

XHTML 1.0 specifies three XML document types that correspond to three DTDs: Strict, Transitional, and Frameset. The most common is XHTML transitional. The `DOCTYPE` declaration at the beginning of the XHTML document specifies which type is being used:

- XHTML 1.0 Strict

  Use this type when you want clean markup, free of presentational clutter. Use this together with Cascading Style Sheets. Example 1-12 shows a strict DTD.

  *Example 1-12 Strict DTD*

  ```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
  ```

- XHTML 1.0 Transitional

  Use this type when you want to take advantage of HTML's presentational features and when you want to support browsers that do not understand Cascading Style Sheets. Example 1-13 shows a transitional DTD.

  *Example 1-13 Transitional DTD*

  ```
  <!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
  ```

- XHTML 1.0 Frameset

  Use this type when you want to use HTML frames to partition the browser window into two or more frames. Example 1-14 shows a frameset DTD.

  *Example 1-14 Frameset DTD*

  ```
  <!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
  ```

### XLink

XML Linking Language (XLink) is a powerful and compact specification for the use of links in XML documents.

Every developer is familiar with the linking capabilities of the web today. However, as the use of XML grows, we quickly realize that simple tags similar to those in Example 1-15 are not going to be enough in the near future.

*Example 1-15 Simple tags*

```
<a href="elem_lessons.html">Freud</a information about X > are not going to be
enough for many of our needs.
```

XLink allows elements to be inserted into XML documents to create and describe links between resources. It uses XML syntax to create structures that can describe links similar to the simple unidirectional hyperlinks of today's HTML, and also more sophisticated links.

XLink provides a framework for creating both basic unidirectional links and more complex linking structures. It allows XML documents to perform the following tasks:

- Assert linking relationships among more than two resources.
- Associate metadata with a link.
- Express links that reside in a location separate from the linked resources.

Although XLink has not been implemented in any of the major commercial browsers yet, its impact will be crucial for the XML applications of the near future. Its extensible and easy-to-learn design can be an advantage as the new generation of XML applications develop.

For more information about Xlink, see the specification document from W3C:

http://www.w3.org/TR/xlink/

### XPointer

XML Pointer Language (XPointer) specifies a language that builds upon the XPath, to support addressing into the internal structures of XML documents. In particular, it provides for specific references to elements, character strings, selections, and other parts of XML documents (whether or not they bear an explicit ID attribute) by using traversals of a document's structure and choice of parts based on their properties, such as element types, attribute values, character content, and relative position, containment, and order. Xpointer defines the meaning of the "selector" or "fragment identifier" portion of URIs that locate resources of text/xml and application/xml MIME media types.

In XPointer, you define the addressing expression to link XML documents using XPath. For more information about XPointer, see the specification documents from W3C:

http://www.w3.org/TR/xptr/

## 1.2.8 XSL, XSLT, Xpath, and XHTML examples

We first look at example stylesheets that use two implementations of XSLT 1.0 and XPath 1.0

Consider the XML file shown in Example 1-16. It is a simple file that we use as the source of information for our XSLT transformation.

*Example 1-16 hello.xm*

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="hello.xsl"?>
<greeting>Hello world.</greeting>
```

Note that the stylesheet association processing instruction in line two refers to a stylesheet with the name `hello.xsl` of type XSL. Recall that an XSLT processor is not obliged to respect the stylesheet association preference, so we first use a standalone XSLT processor with the stylesheet `hellohtm.xsl`, shown in Example 1-17.

*Example 1-17 hellohtm.xsl*

```
<?xml version="1.0"?><!--hellohtm.xsl-->
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xsl:version="1.0">
<head><title>Greeting</title></head>
<body><p>Words of greeting:<br/>
<b><i><u><xsl:value-of select="greeting"/></u></i></b>
</p></body>
</html>
```

This file looks like a simple XHTML file, an XML file that uses the HTML vocabulary. Although it is simply that, we are allowed to inject into the instance the XSLT instructions by using the prefix for the XSLT vocabulary that is declared in line three. We can use any XML file as an XSLT stylesheet, if it declares the XSLT vocabulary within and indicates the version of XSLT being used. Any prefix can be used for XSLT instructions, although convention often sees XSL: as the prefix value.

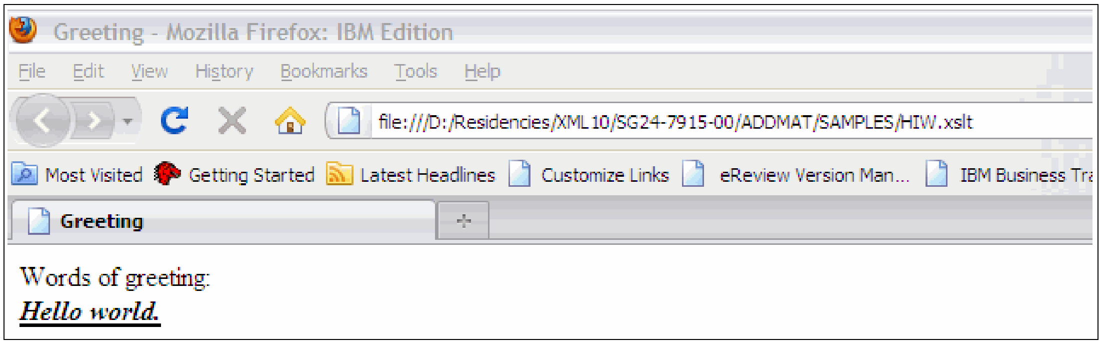
The `xsl:value-of` instruction uses an XPath expression in the `select=` attribute to calculate a string value from our source information. XPath views the source hierarchy using parent/child relationships. The XSLT processor's initial focus is the root of the document, which is considered the parent of the document element. Our XPath expression value "`greeting`" selects the child named "greeting" from the current focus, thus returning the value of the document element named "greeting" from the instance.

We invoke the XSLT processor to point to which is the XML source file, which is the XSL stylesheet, and where to leave the result. Example 1-18 shows the result file.

*Example 1-18 Output from XSLT processor*

```
<html>
<head>
<title>Greeting</title>
</head>
<body>
<p>Words of greeting:<br>
<b><i><u>Hello world.</u></i></b>
</p>
</body>
</html>
```

This is an HTML file; any browser can interpret it as shown in Figure 1-4.



file:///D:/Residencies/XML10/SG24-7915-00/ADDMAT/SAMPLES/HIW.xslt

Words of greeting:
***<u>Hello world.</u>***

*Figure 1-4 Hello world*

## 1.3 What is in this book

In this book, we provide information in four areas that are related to the use of pureXML in a DB2 for z/OS environment:

- Level set information on XML

  We describe XML basics in 1.1, “Importance of XML data” on page 2 and 1.2, “XML introduction” on page 4 of this chapter.

  Chapter 2, “XML and DB2 for z/OS” on page 21 describes pureXML support in DB2 for z/OS.

- Introducing XML in a DB2 for z/OS environment

  Chapter 3, “Application scenario” on page 47 describes the scenario that we use across the various implementation steps.

  Chapter 4, “Creating and adding XML data” on page 53 explains how to define XML objects in a DB2 for z/OS environment.

Chapter 5, “Validating XML data” on page 75 introduces the need for validating documents and the techniques to do so.

Chapter 6, “DB2 SQL/XML programming” on page 89 gives the basis on SQL/XML programming.

- Application development

  We document the steps for the implementation of a simple but meaningful XML application scenario. We provide samples in COBOL and Java language. The purpose is to provide an easy path to follow to integrate the XML data type for the traditional COBOL and DB2 user or the more innovative Java developer.

  Chapter 7, “Using XML with Java” on page 131 describes the Java implementation.

  Chapter 8, “Using XML with COBOL” on page 157 describes the COBOL implementation.

- Database administration

  We also add considerations for the data administrator and suggest best practices for ease of use and better performance.

  Chapter 9, “Utilities with XML” on page 183 revisits most of the DB2 utilities when they are used for XML data type.

  Chapter 10, “XML-related tasks for the DBA” on page 233 highlights differences in administering XML data.

  Chapter 11, “Performance considerations” on page 247 provides a checklist of the major performance considerations when deploying an application that uses pureXML.

# 2

# XML and DB2 for z/OS

In this chapter, we provide a concise overview of the XML capabilities within DB2. The intent of this chapter is to introduce the major elements of pureXML, and to explain how DB2's support for XML data helps to easily store and process XML data in an efficient and productive way.

We also summarize the XML infrastructure within DB2 that is required to support these XML capabilities. Most of the XML function is available for immediate use after a standard DB2 10 for z/OS installation, however several facilities (such as XML schema validation) require that optional parts of the installation process are completed. We clarify these optional steps.

If you want a brief introduction to DB2 pureXML without excessive technical detail, read this chapter. This chapter also acts as a start for users who want to understand the technical details of pureXML, which are covered in more detail in subsequent chapters.

This chapter contains the following topics:

- XML capabilities provided by DB2
- Supporting infrastructure
- Choice of tools

© Copyright IBM Corp. 2011. All rights reserved.

## 2.1 XML capabilities provided by DB2

With DB2, XML documents can be stored, managed, and accessed as first class objects within a DB2 database. XML documents can contain large and flexible data structures. These data structures can be stored in DB2 (which many relational databases allow to some extent), and additionally, the nodes and elements within an XML document can be accessed and indexed to the finest level of granularity.

The “native” support for XML is what makes DB2 pureXML so powerful. The reason is because XML documents and schemas can be used within DB2 with minimal administration work, as we show within the application scenario that is used in this book.

Several main capabilities that are provided by DB2 pureXML are as follows:

- A native XML data type
- SQL/XML language, providing XML functions within the SQL language to access XML structures with the full power of XPath expressions
- Hybrid data access, whereby relational and XML structures can be accessed together using a single SQL/XML statement
- Read and write access to XML documents and sub-documents
- XML Indexes (based on XPath expressions) to provide efficient access paths
- XML schema validation (against an XML schema registered in the Schema Repository) including support for multiple versions of XML schemas

Collectively with these capabilities, you may use DB2 as a repository for both relational and XML data structures. Starting with version 9, DB2 for z/OS is a hybrid database. You can store both kinds of structures in a single database, write applications that use both kinds of structures concurrently, and manage all your data with the same set of database administration utilities.

The XML model of data is significantly different from the relational model of data. Java and JDBC provide many powerful XML manipulation capabilities.

COBOL and PL/I have also added many XML manipulation capabilities, as described at the following addresses:

- http://www.ibm.com/support/docview.wss?uid=swg27004198&aid=1
- http://www.ibm.com/software/awdtools/cobol/zos/

The range of these XML capabilities that are provided by DB2 helps to easily incorporate XML data into all DB2 databases and applications, including those written in languages like COBOL and PL/I.

This section reviews the major XML capabilities of DB2, and explains why each of these capabilities is important for building DB2 applications that contain XML data.

We examine the following topics:

- Native XML data type
- SQL/XML language
- Hybrid data access
- XML update
- XML indexes
- XML schema repository and schema validation

## 2.1.1 Native XML data type

Although all data types are equal, some are more equal than others. To think of XML as a data type, equivalent in scope and importance to an integer for example, is misleading. XML is implemented as a data type within DB2, but it also encompasses an entire data model that contains many other data types (including the integer).

Given the proliferation of XML within modern systems, a database must be able to store XML documents efficiently, and support efficient data access to any data element within the XML documents (with all the other qualities of services that a database must provide).

Several vendors have provided XML-only databases for native XML repository and search engine. Other relational database vendors have provided ways of storing XML documents, and “stripping” out important data elements into relational structures (such as DB2 V7 and V8 did).

The XML data type is part of the ANSI SQL standard. DB2 implements this standard, and additionally provides constructs such as XML indexes to make it productive and performant.

What DB2 pureXML provides is the combination of native XML data support with its established relational data support, in a fully integrated way. The XML data type (first introduced in DB2 9) is the most fundamental component of DB2’s support for XML. It is different from storing XML as a string data type (as was done in DB2 V7 and V8).

### Before the native XML data type existed

When XML was stored as a string (in DB2 V8 and earlier), DB2 had no inherent “understanding” of the structure within the XML document. DB2 V7 provided the XML Extender facility, which helped you to parse an XML document that was stored in DB2 as a string, and optionally strip out XML elements of interest. However, you had to perform this parsing every time you wanted to access an XML document. If you wanted to process the data within the stored XML documents, you had two choices:

- Read and parse the entire document to access the data elements within the XML documents. This approach can be expensive, particularly if you are processing many rows through a cursor operation.
- Perform much up-front database administration and development work to strip out the data elements that might be used for searching and joining, and store them in additional DB2 columns with traditional DB2 data types.

The first approach is not practically viable from a performance or CPU-cost perspective. XML parsing is a CPU-intensive activity that you do not want to incur each time you access the XML data.

The second approach is practical, but is unproductive. This approach requires a significant development project to be undertaken to prepare the DB2 database before you can start to develop the application that will use the XML data.

### After the native XML data type was introduced

The native XML data type (provided by DB2 9 and further improved by DB2 10) allows XML documents to be stored within DB2 in a practical and productive way:

- Performance: The XML document is parsed once only, as it is inserted (or loaded) into DB2. The internal structure of the XML document is then accessible (without being re-parsed) by the SQL/XML functions that are provided by DB2.
- Productivity: No stripping of XML elements into separate DB2 columns is required (although if you choose to store various XML data elements in DB2 columns for database design reasons, do so is easy). All the XML elements can be referenced and indexed where they reside within the XML column.

XML documents are placed inside DB2 by inserting them into a column that has been defined with the XML data type, as illustrated in Example 2-1.

*Example 2-1 Defining and populating an XML column*

```
CREATE TABLE XMLR3.XMLTEST_TAB (
   TESTKEY BIGINT,
   TESTXMLCOL XML )
      IN XMLR3DB.TEST_TS ;

INSERT INTO XMLR3.XMLTEST_TAB ( TESTKEY, TESTXMLCOL )
   VALUES ( 1, '<customerinfo><name>Amir Malik</name>
            <phone type="work">408-555-1358</phone></customerinfo>') ;
```

The next sections describe how XML data can be processed after it is inside a DB2 table, stored in a column of the XML data type.

## 2.1.2 SQL/XML language

Now that you can store XML as a native data type within DB2, you need a data access language to work with it.

Although standard SQL is able to access tables with XML columns, it does not have the direct manipulation capabilities to do anything meaningful with the XML structure within the retrieved XML document. For this reason, DB2 implemented SQL/XML extensions to SQL, to provide a range of functions that allow the contents of XML documents to be processed directly. These functions can also be encapsulated in DB2- stored procedures and user-defined functions, so that standard SQL can be used against XML data with functions and procedures that were developed using SQL/XML extensions.

SQL/XML is an extension to the SQL standard as defined by ISO/IEC 9075-14:2003. The SQL/XML part of DB2 pureXML, being standards-based, is supported by many other databases. It provides a range of XML related extensions to the SQL language, to allow XML data to be accessed. The functions provided by SQL/XML can be categorized into the following main groups.

- XML publishing functions, which allow XML documents to be created from the contents of relational data
- XML handling functions, which allow the user to embed XPath expressions in SQL statements. XPath expressions are a subset of the XQuery standard.
- XML conversion functions, which support the interchange of data between relational and XML models of data

### XML publishing functions

SQL/XML provides a range of XML publishing functions that allow XML documents to be created from the contents of relational data. Example 2-2 shows a simple example of XML publishing functions that conveys the concepts of what is possible. In this example, we create a normal relational table, populate it, and then generate an XML document based on the contents of the relational table.

*Example 2-2 XML Publishing Functions of SQL/XML*

```
Create Table Address (
   CUSTNAME    VARCHAR(70),
   STRTNM      VARCHAR(70),
   BLDGNB      VARCHAR(16),
   PSTCD       VARCHAR(16),
   TWNNM       VARCHAR(35)
)  ;

Insert into Address
   values ('John Smith', 'Bailey Avenue', '555', '95141', 'San Jose') ;

SELECT
   XMLELEMENT(
      NAME "MsgRcpt",
      XMLELEMENT(NAME "Nm", CUSTNAME),
      XMLELEMENT(NAME "PstlAdr",
         XMLELEMENT(NAME "StrtNm", STRTNM),
         XMLELEMENT(NAME "BldgNb", BLDGNB),
         XMLELEMENT(NAME "PstCd", PSTCD),
         XMLELEMENT(NAME "TwnNm", TWNNM)
) ) from Address ;

-- yields result

<MsgRcpt>
   <Nm>John Smith</Nm>
   <PstlAdr>
      <StrtNm>Bailey Avenue</StrtNm>
      <BldgNb>555</BldgNb>
      <PstCd>95141</PstCd>
      <TwnNm>San Jose</TwnNm>
   </PstlAdr>
</MsgRcpt>
```

XML publishing functions are straightforward. They provide a range of string concatenation steps that can be combined together to form an XML documents. They simplify what you could already have done with the standard SQL CONCAT function.

## XML handling functions

The XML handling functions are what enables SQL statements to access and manipulate the contents of XML documents stored within XML documents held within DB2. They include XMLQUERY, XMLTABLE and XMLEXISTS functions. Each of these functions makes use of XPath expressions, which is a subset of XQuery.

Existing DB2 programmers can extend their skills to include XPath expressions, to extend the relational programs that they already write, and to incorporate XML data. Their capabilities are best explained with several simple examples, based on a simple table with two columns, illustrated in Example 2-3.

*Example 2-3 Table with XML column*

```
Create Table XMLADDRESS (
   XID INT,
   XMLADDRESS XML ) ;

Insert into XMLADDRESS ( XID , XMLADDRESS )
   values (1, '<MsgRcpt><Nm>John Smith</Nm><PstlAdr>
         <StrtNm>Bailey Avenue</StrtNm><BldgNb>555</BldgNb>
         <PstCd>95141</PstCd><TwnNm>San Jose</TwnNm></PstlAdr></MsgRcpt>' ) ;

Insert into XMLADDRESS ( XID , XMLADDRESS )
   values (2, '<MsgRcpt><Nm>John Doe</Nm><PstlAdr>
         <StrtNm>Antelope Street</StrtNm><BldgNb>32</BldgNb>
         <PstCd>87233</PstCd><TwnNm>San Diego</TwnNm></PstlAdr></MsgRcpt>' ) ;
```

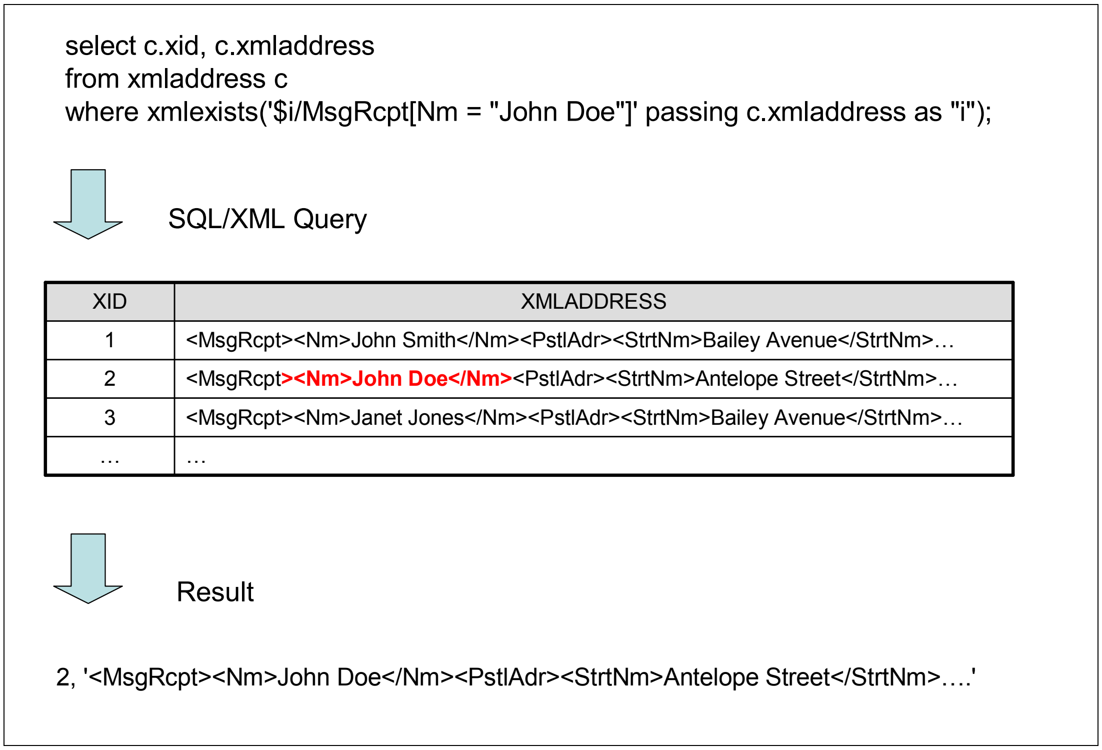
### *XMLEXISTS*

The XMLEXISTS function is used as a predicate to retrieve DB2 rows, based on predicates that are applied to the data within an XML column, as illustrated in Example 2-4. In this example, each XML document is examined to see whether an `<Nm>` data element exists with a value of "`John Doe`" at `/MsgRcpt` XPath location, and returns the rows where the predicate is satisfied.

*Example 2-4 Simple XMLEXISTS*

```
select c.xid, c.xmladdress
   from   xmladdress c
   where  xmlexists('$i/MsgRcpt[Nm = "John Doe"]'
      passing c.xmladdress as "i");

--yields result

XID XMLADDRESS
--- ------------
  2 <MsgRcpt><Nm>John Doe</Nm><PstlAdr><StrtNm>Antelope...
```

Figure 2-1 might help you to visualize the query.



```
select c.xid, c.xmladdress
from xmladdress c
where xmlexists('$i/MsgRcpt[Nm = "John Doe"]' passing c.xmladdress as "i");
```

| XID | XMLADDRESS |
|---|---|
| 1 | <MsgRcpt><Nm>John Smith</Nm><PstlAdr><StrtNm>Bailey Avenue</StrtNm>… |
| 2 | <MsgRcpt>**<Nm>John Doe</Nm>**<PstlAdr><StrtNm>Antelope Street</StrtNm>… |
| 3 | <MsgRcpt><Nm>Janet Jones</Nm><PstlAdr><StrtNm>Bailey Avenue</StrtNm>… |
| … | … |

```
2, '<MsgRcpt><Nm>John Doe</Nm><PstlAdr><StrtNm>Antelope Street</StrtNm>….'
```

*Figure 2-1 SQL/XML query with XMLEXISTS predicate*

Let us examine the structure of the XMLEXISTS predicate as the first example in this book of the SQL/XML extensions to the SQL language:

- The XMLEXISTS clause is the same as any other relational predicate, except that the XMLEXISTS clause contains XML syntax expressions.
- If the `Nm` field had been a VARCHAR column, the expression would have been

  `where Nm = "John Doe"`
- The XPath expression is applied to the XML column using the `passing` clause. In this way, the XML document in each row that is accessed is passed as **“`i`”** to the XPath predicate for evaluation
- The comparison that is made by the XMLEXISTS clause is identical to what would have been done relationally. The contents of the XML location `/MsgRcpt/Nm` are compared to the `"John Doe"` string value:
  - If the values match, the predicate is satisfied and the row qualifies.
  - If the values do not match, the row does not qualify.
  - If the data types do not match, the row does not qualify.

Note that the SQL/XML statement does not know whether the contents at `/MsgRcpt/Nm` is a string or a number or any other data type. It determines this at run time. The data type of the `/MSgRcpt/Nm` location may constrained to be a particular type if the document conforms to an XML schema. XML schemas and validation of XML documents against XML schemas is covered in 2.1.6, “XML schema repository and schema validation” on page 36.

Also, be aware of namespaces. Normally, XML document have a declared namespace. A namespace provides the ability to uniquely define a data element or attribute within an XML document, so that when a tag such as <phone> is used more than once in an XML document, the precise context of that <phone> tag is understood.

If an XML document has a namespace declaration, the query must also define the namespace, to ensure that we are referencing the correct XML data. Example 2-5 shows how a namespace is declared on the XMLEXISTS function, if a namespace is required.

*Example 2-5 XMLEXISTS with namespace declaration*

```
select c.xid, c.xmladdress
   from xmladdress c
   where xmlexists('
      declare default element namespace “http:\\www.names.com";
      $i/MsgRcpt[Nm = "John Doe"]'
      passing c.xmladdress as "i");
```

By coding the following statement, the DBA or developer must consider whether or not the name column must be indexed to avoid a table space scan:

```
select * from table where name = ‘John Doe’
```

Exactly the same consideration applies to XML data. If we really want to code the SQL/XML statement in Example 2-4 on page 26, we must consider an XML index to provide an efficient access path to the <Nm> elements in the XML documents. Otherwise, as a result, we will be executing the XPath expression against every single XML document in the table.

Of course, the DB2 optimizer is able to choose an access path based on a combination of XML and relational predicates. Therefore, relational and XML indexes are both available for access path selection, and can both be used in the same access plan.

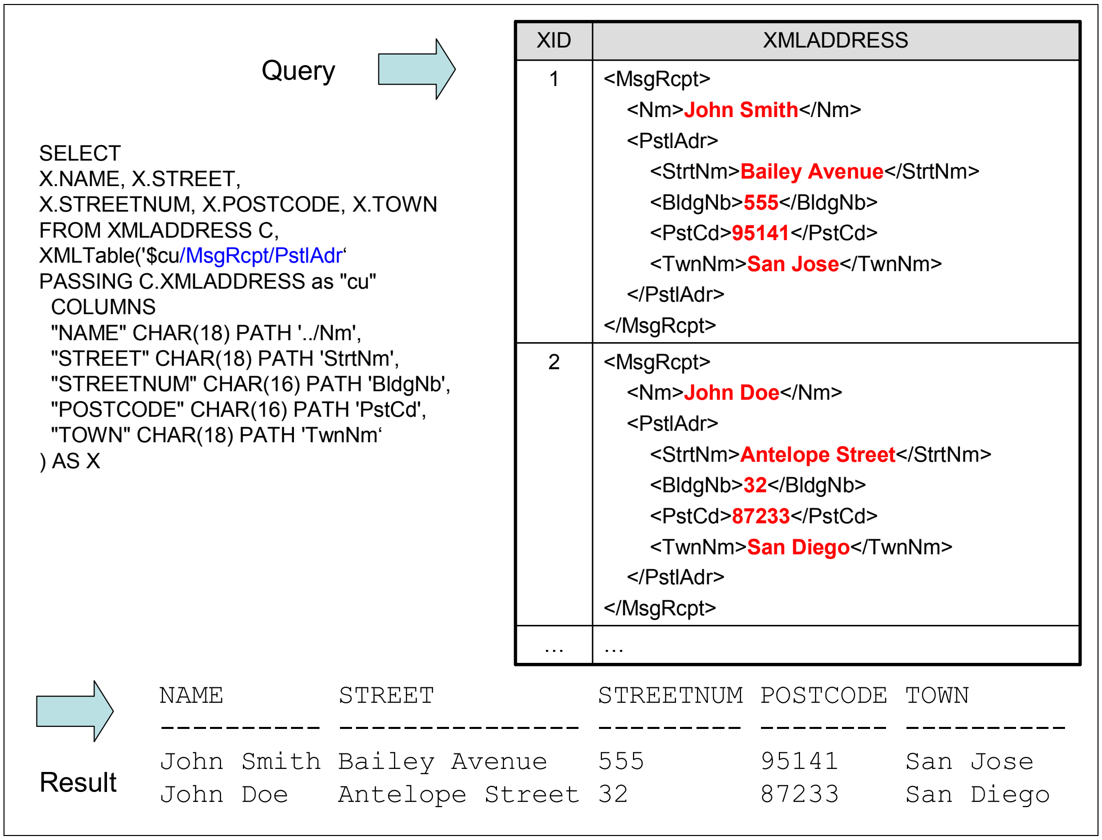
### XMLTABLE

The XMLTABLE function is used to retrieve XML elements and attributes from an XML document and map them to a relational table structure, to be used by programs exactly as though the data had been retrieved from a wholly relational table, as illustrated in Example 2-6.

*Example 2-6 Simple XMLTABLE*

```
SELECT X.NAME, X.STREET, X.STREETNUM, X.POSTCODE, X.TOWN
   FROM XMLADDRESS C,
   XMLTable('$cu/MsgRcpt/PstlAdr'
      PASSING C.XMLADDRESS as "cu"
         COLUMNS
            "NAME"        CHAR(18)  PATH '../Nm',
            "STREET"      CHAR(18)  PATH 'StrtNm',
            "STREETNUM"   CHAR(16)  PATH 'BldgNb',
            "POSTCODE"    CHAR(16)  PATH 'PstCd',
            "TOWN"        CHAR(18)  PATH 'TwnNm'
   ) AS X

... yields

NAME       STREET             STREETNUM        POSTCODE         TOWN
---------- ------------------ ---------------- ---------------- ----------
John Smith Bailey Avenue      555              95141            San Jose
John Doe   Antelope Street     32              87233            San Diego
```

The XMLTABLE in Example 2-6 on page 28 shows how individual elements within an XML document can be mapped to a relational structure, and retrieved alongside other relational columns from the DB2 table which the XML column belongs to. Figure 2-2 provides a graphical representation of the XMLTABLE function.



```
SELECT
X.NAME, X.STREET,
X.STREETNUM, X.POSTCODE, X.TOWN
FROM XMLADDRESS C,
XMLTable('$cu/MsgRcpt/PstlAdr‘
PASSING C.XMLADDRESS as "cu"
  COLUMNS
  "NAME" CHAR(18) PATH '../Nm',
  "STREET" CHAR(18) PATH 'StrtNm',
  "STREETNUM" CHAR(16) PATH 'BldgNb',
  "POSTCODE" CHAR(16) PATH 'PstCd',
  "TOWN" CHAR(18) PATH 'TwnNm‘
) AS X
```

| XID | XMLADDRESS |
|---|---|
| 1 | <MsgRcpt><br><Nm>**John Smith**</Nm><br><PstlAdr><br><StrtNm>**Bailey Avenue**</StrtNm><br><BldgNb>**555**</BldgNb><br><PstCd>**95141**</PstCd><br><TwnNm>**San Jose**</TwnNm><br></PstlAdr><br></MsgRcpt> |
| 2 | <MsgRcpt><br><Nm>**John Doe**</Nm><br><PstlAdr><br><StrtNm>**Antelope Street**</StrtNm><br><BldgNb>**32**</BldgNb><br><PstCd>**87233**</PstCd><br><TwnNm>**San Diego**</TwnNm><br></PstlAdr><br></MsgRcpt> |
| … | … |

```
NAME       STREET          STREETNUM POSTCODE TOWN
---------- --------------- --------- -------- ----------
John Smith Bailey Avenue   555       95141    San Jose
John Doe   Antelope Street 32        87233    San Diego
```

*Figure 2-2 XMLTABLE function example*

The mapping is based on an XPath expression that is used to navigate to a particular anchor point in the XML document (`/MsgRcpt/PstlAdr`) and then using relative XPath expressions (such as `/StrtNm` at the next level down, and `../Nm` at the next level up from that anchor point) to access XML data elements that are mapped to relational fields (such as `STREET VARCHAR(18)`).

Note that the example in Figure 2-2 has mapped data from `Nm`, `StrtNm`, and `TwnNm` as `CHAR(18)`, whereas the original source of the data for all three fields was `VARCHAR(70)` in Example 2-2 on page 25. The XMLTABLE function will perform a range of standard data type casting, and will return an error if the casting is not possible. For example, if you attempted to cast Nm as an integer, you would get SQL16061N. The value "`John Smith`" cannot be constructed as, or cast (using an implicit or explicit cast) to the "`xs:integer`" data type.

Later in this book, more details are provided for managing the XML to relational mapping and handling errors that might arise (for example, when casting an XML element to a specific relational data type).

### *XMLQUERY*

The XMLQUERY function is used to embed XPath expressions within an SQL/XML statement. It always produces a column of type XML, and, as a scalar function, it returns a sequence of items for each document (row), as illustrated in Example 2-7 on page 30.

*Example 2-7 Simple XMLQUERY*

```
select
   xmlquery('$i/MsgRcpt/Nm ' passing c.xmladdress as "i")
   as Names
   from xmladdress c;

-- yields

Names
-------------------
<Nm>John Smith</Nm>
<Nm>John Doe</Nm>
```

The XMLQUERY in Example 2-7 shows how XPath expressions can be executed within an SQL statement, with the resulting XML document (or subdocument) being returned as an XML structure. This particular example retrieves the XML structure at node `/MsgRcpt/Nm` for every single XML document within the XMLADDRESS table.

The XMLEXISTS and XMLTABLE functions are particularly attractive when using traditional programming languages, because they return data in a relational format. XMLTABLE can also return an XML column.

The XMLQUERY function differs slightly because it returns a scalar value XML type, not necessarily an XML document (usually it is not a document). This difference does not necessarily introduce extra complexity to the traditional programmer, because the returned XML type can be cast to a string data type, or moved on to another repository such as WebSphere® Message Queue (MQ).

### *Summary of XML handling functions*

Collectively, the XMLEXISTS, XMLQUERY, and XMLTABLE functions is what the traditional mainframe programmer can use to access and manipulate XML documents with a minimal increase in skills required (namely, the ability to write XQuery and XPath expressions, and embed them in SQL statements). We examine the considerations for using each of these functions with index access in Chapter 11, “Performance considerations” on page 247.

## XML conversion functions

The XML conversion functions support the interchange of data between relational and XML models of data. The XML conversion functions are XMLCAST, XMLPARSE and XMLSERIALIZE.

XMLCAST function casts the contents of an XML data element to a relational data type as shown in Example 2-8.

*Example 2-8 XMLCAST casting a numeric data element to varchar or integer*

```
select
   xmlcast(xmlquery('$i/MsgRcpt/PstlAdr/BldgNb'
      passing c.xmladdress as "i") as varchar(10))
   as streetnumber from xmladdress c;

select
   xmlcast(xmlquery('$i/MsgRcpt/PstlAdr/BldgNb'
      passing c.xmladdress as "i") as integer)
   as streetnumber from xmladdress c;
```

XMLPARSE function parses a string of a well-formed XML document that conforms to XML 1.0 and returns an XML type. IXMLPARSE can be used in isolation or as part of an SQL INSERT operation as shown in Example 2-9. XMLPARSE also provides the option to strip white space, for storage efficiency, or preserve it.

*Example 2-9 XMLPARSE function and white space handling*

```
INSERT INTO XMLDEMO1 VALUES(201,
   XMLPARSE( DOCUMENT
   '<emails><email emailUse="work">       fred_smith@uk.ibm.com </email>
      </emails>'
   PRESERVE WHITESPACE ) );


INSERT INTO XMLDEMO1 VALUES(202,
   XMLPARSE( DOCUMENT
   '<emails><email emailUse="work">       fred_smith@uk.ibm.com </email>
      </emails>'
   STRIP WHITESPACE ) );
```

XMLSERIALIZE function converts an XML document into a serialized string value. Example 2-10 converts an XML document into a CLOB in UTF-8. The resulting data type can also be BLOB, DBCLOB, and others.

*Example 2-10 XMLSERIALISZE example to convert an XML document to a UTF-8 CLOB*

```
SELECT e.xid, XMLSERIALIZE(XMLADDRESS AS CLOB) AS "result"
   from xmladdress  e;
```

### 2.1.3 Hybrid data access

One of the big advantages of adding XML support to DB2 is that hybrid database applications can be constructed, combining the relational model of data and the XML model of data.

A good example of a hybrid data application might be insurance quotes. When an Internet user fills in multiple HTML forms to get an insurance quote, the insurance company wants to persist that information in a database, so that the Internet user can return at a later time to purchase the policy that was quoted.

XML can be an excellent data model for storing insurance quotes because the data can typically contain a large number of data elements, and various quotes may be structured in many various ways. For example, the HTML forms that are filled in for a single-driver car-insurance policy for a small vehicle can differ significantly from the HTML forms that are filled in by a married couple insuring a powerful sports utility vehicle, and adding their 18 year-old son as a named driver:

- If the quotation application was implemented in a relational structure, 10 or 20 main tables, and multiple code tables, might exist.
- If the quotation application was implemented in an XML structure, a single XML document can contain all the structure and constraints for the quotes.

However, the existing client and policy systems likely will have been written many years ago using the relational model of data. Therefore, the new quotes application must be able to bridge the gap between XML and relational data models.

DB2's support for a hybrid data model can help to more easily develop a new Internet quotes application. A possibility is for the Internet quotes application to store the quote as an XML document in a DB2 table. If the quote is subsequently taken up by the client, then the information stored within the XML document containing the quote details can be retrieved by using SQL/XML and re-used as input to the relational tables of the client and policy systems.

The XMLTABLE function provides the basis for a simple use case. The address details can be read from an XML document that contains the quote, and inserted into an address table within the policy system, as shown in Example 2-11.

*Example 2-11 Hybrid data access*

```
INSERT INTO POLICY_SYSTEM.ADDRESS_TABLE ( ADDR1, TOWN, POSTCODE )
   SELECT X.ADDRESS, X.TOWN, X.POSTCODE
   FROM
      QUOTE_SYSTEM.XMLQUOTE_TABLE C,
      XMLTable('$cu/quote/addresses/address'
      PASSING C.CUSTCONTACTS as "cu"
         COLUMNS
            "ADDRESS"    CHAR(20)  PATH 'addressLine1',
            "TOWN"  CHAR(20)  PATH 'addressPostTown',
            "POSTCODE"    CHAR(20)  PATH 'addressPostCode'
      ) AS X
   WHERE X.QUOTE_ID = 123456 ;
```

Clearly, this simplistic example does not reflect the way that insurance systems might be designed, with error-checking, input validation, and so on. However, the example still serves to illustrate that the DB2 hybrid data model does provide an integrated database platform upon which bridging the gap between relational and hierarchical models of data can be easier.

## 2.1.4 XML update

Full read and write access is provided for XML documents within DB2.

Inserting and updating XML data is supported with the XMLPARSE option, which parses a string value to return an XML document, and optionally strips or preserves white space. The white space handling is illustrated in Example 2-9 on page 31.

Updating XML documents can be performed by either replacing the whole document, or updating a part of it. A full replacement of an XML document is coded with a simple SQL update statement, and an XMLPARSE function call if the source data is in string format. A partial document update uses the XMLMODIFY function to change an existing XML document.

The XMLMODIFY function depends on the underlying table space being a universal table space, so that multiple XML versions are supported. Support for multiple XML versions is implemented in DB2 10 to improve concurrency and reduce locking: A new version of the XML document is created during an update, which allows SQL read operations (with the appropriate isolation level) to access the old version of the XML document concurrently.

The XMLMODIFY function operates on a node within an XML document, and has three usage variations:

- Replace a node
- Delete a node
- Insert a node

This section shows examples of all three variations of XMLMODIFY, based on the initial table contents shown in Example 2-12.

*Example 2-12 Initial contents of XMLADDRESS table*

```
XID XMLADDRESS
--- ------------------------------------
  2 <MsgRcpt>
       <Nm>John Doe</Nm>
       <PstlAdr>
          <StrtNm>Antelope Street</StrtNm>
          <BldgNb>32</BldgNb>
          <PstCd>87233</PstCd>
          <TwnNm>San Diego</TwnNm>
       </PstlAdr>
    </MsgRcpt>
```

## XMLMODIFY to replace a node

The first example (Example 2-13) replaces the `/MsgRcpt/Nm` node with another XML fragment. The replacement XML fragment is a literal string value, which is parsed using the XMLPARSE function.

*Example 2-13 XMLMODIFY to replace a node*

```
UPDATE XMLR3.XMLADDRESS C
   SET C.XMLADDRESS = XMLMODIFY(
      'replace node /MsgRcpt/Nm with $x',
      XMLPARSE('<Nm>Johnny Doe</Nm>') AS "x")
   where C.XID = 2

-- changes the second row in the XMLADDRESS table to the following:

XID XMLADDRESS
--- ------------------------------------
  2 <MsgRcpt>
       <Nm>Johnny Doe</Nm>
       <PstlAdr>
          <StrtNm>Antelope Street</StrtNm>
          <BldgNb>32</BldgNb>
          <PstCd>87233</PstCd>
          <TwnNm>San Diego</TwnNm>
       </PstlAdr>
    </MsgRcpt>
```

## XMLMODIFY to delete a node

The second example (Example 2-14) deletes the `<PstlAdr>` node from the document.

*Example 2-14 XMLMODIFY to delete a node*

```
UPDATE XMLR3.XMLADDRESS C
   SET C.XMLADDRESS = XMLMODIFY(
      'delete node /MsgRcpt/PstlAdr')
   WHERE C.XID  = 2

-- changes the second row in the XMLADDRESS table to the following:

XID XMLADDRESS
--- ------------------------------------
  2 <MsgRcpt>
       <Nm>Johnny Doe</Nm>
    </MsgRcpt>
```

## XMLMODIFY to insert a node

The third example (Example 2-15) inserts the `<Notes>` node as the last child node at XPath `/MsgId` location.

*Example 2-15 XMLMODIFY to insert a node*

```
UPDATE XMLR3.XMLADDRESS C
   SET C.XMLADDRESS = XMLMODIFY(
      'insert node $x as last into /MsgRcpt',
      XMLPARSE('<Notes>Testing</Notes>') AS "x")
   where C.XID = 2

changes the second row in the XMLADDRESS table to the following:

XID XMLADDRESS
--- ------------------------------------
  2 <MsgRcpt>
       <Nm>Johnny Doe</Nm>
       <Notes>Testing</Notes>
    </MsgRcpt>
```

In all cases, the following XML schema validation considerations apply:

- If the table has an XML type modifier for the INFO column, then the resultant XML value will be checked for well formedness and will also be checked for conformance to the registered XML schema.
- If the table does not have an XML type modifier for the INFO column, then the resultant XML value will be checked for well formedness only. XML schema validation could be requested manually, by calling the DSN_XMLVALIDATE function.

Sub-document update becomes important for performance reasons if small changes need to be made to large XML documents. For example, changing the details of one book in an XML document that contains a book catalog would be a lot cheaper than replacing and revalidating the whole document.

### 2.1.5 XML indexes

XML indexes are as important to DB2 as relational indexes are. The best way to introduce XML indexes is to start with the similarities to relational indexes, and then to identify the differences.

The major reasons for defining relational indexes in DB2 are as follows:

- Efficient access path to data (minimizing I/O and CPU resource consumption)
- Option to apply a unique constraint

These same reasons also apply to XML indexes. Their fundamental purposes are to facilitate performance and to enforce uniqueness.

However, the nature of an XML index has several differences from a relational index. The major differences are as follows:

- Relational indexes may be defined on one or more relational columns. XML Indexes may be defined only on one XML element or XML attribute (using an XML pattern expression).
- Relational indexes always have one index entry for every row in a table. However, XML indexes are much less prescriptive. XML indexes are based on an XML pattern. An XML pattern may occur any number of times in an XML document. Therefore, XML indexes may contain 0, 1, or many entries for each row in the table.
- Relational indexes are always based on the data types of the column or columns on which they are defined. The data types found at the locations of an XML pattern may be many and varied unless an appropriate XML schema is enforced. Using XML schema validation against a well-defined XML schema will allow the data types of XML elements to be controlled (as covered in 2.1.6, “XML schema repository and schema validation” on page 36).
- Relational indexes can be used to support table clustering. XML indexes may not be used for table clustering support.

XML indexes have a more complex set of physical design considerations than relational indexes. A good understanding of XPath expressions and XML schemas and XML namespaces is essential to designing XML indexes that are likely to be chosen by the DB2 optimizer.

Having sounded a caution that XML index design requires careful consideration of a number of new factors, we also want to state how powerful and productive they are. Without XML indexes, a database application would have to go through an extra development phase where key data elements are stripped to relational tables, and indexed by using traditional relational indexes. With XML indexes, you can avoid this extra development phase and index your XML data as efficiently as you index your relational data, and realize the simplicity that DB2’s hybrid data model provides.

Example 2-16 provides simple example of how XML indexes are created in DB2.

*Example 2-16 XML Index creation*

```
CREATE INDEX XMLCITY ON XMLDEMO1(CUSTCONTACTS)
   GENERATE KEY USING XMLPATTERN
      '/employeeContacts/addresses/address/AddressPostCode'
   AS SQL VARCHAR(30);
```

In this example, the contents of `'/employeeContacts/addresses/address/AddressPostCode'` XPath location are cast as a `VARCHAR(30)` data type, and an XML index is built:

- If the data type casting is successful, an index entry is created.
- If the XPath does not exist, no index entry is created.
- If the data type casting is unsuccessful, the following occurrences happen:
  - For numeric, date, and time-stamp index data types, error-tolerance applies.
  - For VARCHAR indexes, where the failure of casting is due to length, either CREATE INDEX will fail or INSERT will fail.

An XML index can point to a small number of entries (or none) if the XML pattern of the index does not fit the data.

XML index design requires the nature of XML indexes to be understood, and the matching success of the index to be checked.

The DB2 optimizer can use XML indexes and traditional DB2 indexes together to produce the most efficient access paths for hybrid data access. The way that XML and relational indexes work together is one of the best illustrations of how DB2 makes hybrid data both valuable and performant.

## 2.1.6 XML schema repository and schema validation

DB2 provides many features that allow the database administrator to implement and enforce a relational data model. Unique indexes, primary and foreign keys, check constraints, referential constraints, and so on.

DB2 also provides the ability to implement and enforce an XML data model, through its ability to validate XML documents against a registered XML schema. XML data structures can be defined through XML schemas. An XML schema defines the structure of an XML document by defining the following items:

- The elements that can appear in a document
- The attributes that can appear in a document
- Which elements are child elements
- The order of child elements
- The number of child elements
- Whether an element is empty or can include text
- The data types for elements and attributes
- The default and fixed values for elements and attributes

A simple XML schema is shown in Example 2-17 on page 37, which defines how email information about an individual should be stored. The `<emails>` element can be a parent to 1 - 5 specific `<email>` elements. Each email element is required to have an emailUse tag, which describes the nature of that particular email account (work, home, and so on).

*Example 2-17 Simple XML schema example*

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="emails" maxOccurs="1" minOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="email" maxOccurs="5" minOccurs="1">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute name="emailUse" use="required">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="home"/>
                    <xs:enumeration value="work"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

The XML document in Example 2-18 conforms to the XML schema that is described in Example 2-17.

*Example 2-18 XML document that conforms to previous XML schema*

```
<emails>
  <email emailUse="work">UK000003@uk.ibm.com</email>
  <email emailUse="home">julie.woollacot@isp.com</email>
</emails>
```

DB2 support for XML schema validation consists of the following items:

- An XML schema repository, where XML schemas can be stored within DB2
- A schema validation function, which allows an XML data type to be validated against a schema within the repository
- A Data Definition Language (DDL) option to define an XML column with a type modifier, so that schema validation is enforced by DB2 automatically for that column

The schema repository is populated by using the four XML schema repository stored procedures that are provided by DB2. These stored procedures are invoked by commands from z/OS UNIX® System Services or DB2 for z/OS[1] and DB2 for Linux®, UNIX, and Windows® command line processor (CLP), as shown in the following examples.

The script in Example 2-19 on page 38 shows a schema being registered to the DB2 XML schema repository, and then being completed.

[1] See DB2 for z/OS CLP: http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.apsg/db2z_runspfromclp.htm

*Example 2-19 XML schema registration*

```
register xmlschema http://www.mymodel
   from file://C:\SCHEMAPATH\myschema.xsd
   as SYSXSR.MYXMLSCHEMA ;

complete xmlschema SYSXSR.MYXMLSCHEMA ;
```

The SQL statement in Example 2-20 shows the DSN_XMLVALIDATE function being invoked manually, as part of an INSERT operation to a table (TABLE1).

*Example 2-20 SYSXSR.DSN_XMLVALIDATE example*

```
INSERT INTO TABLE1(XMLCOL1)
   VALUES (SYSIBM.DSN_XMLVALIDATE(:xmldoc, 'SYSXSR.MYXMLSCHEMA')
   );
```

The DDL statement in Example 2-21 shows another Table (TABLE2) being defined, where all XML documents stored in the XML column of that table will be automatically validated.

*Example 2-21 XML type modifier*

```
CREATE TABLE XMLR3.TABLE2 (
   XMLCOL1 XML(XMLSCHEMA ID SYSXSR.MYXMLSCHEMA))
   IN XMLR3DB.XMLR3TS ;
```

The DB2 XML schema validation services are essential for enforcing the integrity of XML documents against the rules that they should conform to. They can be simple to use, and allow complex XML schemas to be incorporated easily, as shown by the ISO 20022 examples later in this book.

## 2.2 Supporting infrastructure

Little extra customization must be performed during a DB2 installation to take advantage of the pureXML capability within DB2. The XML data type is a standard DB2 data type that are ready for immediate use with DB2.

XML validation is an optional facility requiring the XML Schema Repository (XSR). You might not want to use DB2's XML schema validation for documents that have already been validated outside of DB2. However, in general, you must have access to this facility or an equivalent one when you use DB2 pureXML.

For the most current maintenance level, see information APAR II14426. It contains a summary and pointers to all the XML support delivery APARs.

### 2.2.1 XSR installation steps

XSR has a prerequisite that the following software is installed and configured:

- Workload Manager for z/OS (WLM)
- z/OS XML System Services
- Java 2 Technology Edition, V5 or later, 31-bit version
- IBM Data Server Driver for JDBC and SQLJ

XSR consists of four DB2-supplied stored procedures, one DB2 database, five table spaces, eight tables and 13 indexes. Installation job DSNTIJRT is designed to create all DB-supplied stored procedures including these objects.

The stored procedures are listed in Table 2-1.

*Table 2-1 Stored procedures*

| Stored procedure | Description |
|---|---|
| SYSPROC.XSR_REGISTER | C-stored procedure that registers an XML schema as a primary schema document in the DB2 XML schema repository |
| SYSPROC.XSR_ADDSCHEMADOC | C-stored procedure that registers an additional XML schema document to an XML schema in the DB2 XML schema repository |
| SYSPROC.XSR_COMPLETE | Java-stored procedure that completes the registration process of an XML schema |
| SYSPROC.XSR_REMOVE | C-stored procedure that removes a registered XML schema from the XML schema repository |

The installation steps of the XML Schema Repository and functions are as follows:

1. Customize and run the DSNTIJRW and DSNTIJMV jobs:
   - Define the WLM environment and startup procedure for the C language XML schema repository-stored procedures.

     A dedicated WLM environment and startup procedure is required for the XSR-stored procedures that are written in C (XSR_ADDSCHEMADOC, XSR_REGISTER, and XSR_REMOVE). Note the following information:

     - The DSNTIJRW Installation job installs and configures a WLM environment with the default name of DSNWLM_XML, which you can use it to run the XML schema repository stored procedures.
     - The DSNTIJMV installation job installs a WLM startup procedure named ssnmWLMX for that WLM environment.
   - Define the WLM environment and startup procedure for the Java language XML schema repository stored procedure. A dedicated WLM environment and startup procedure is required for the XSR stored procedure written in Java (XSR_COMPLETE). Note the following information:
     - The DSNTIJRW installation job installs and configures a WLM environment with the default name of DSNWLM_JAVA, which you can use it to run the XML schema repository stored procedures.
     - The DSNTIJMV installation job installs a WLM startup procedure named ssnmWLMJ for that WLM environment.
2. Customize and run job DSNTIJRT.

   Run installation Job DSNTIJRT to create the XSR objects (database, table spaces, tables, indexes, stored procedures) and bind the DB2 XSR packages for the stored procedures.
3. Bind the Universal JDBC Driver.

   Bind the packages for the IBM Data Server Driver for JDBC and SQLJ (on UNIX System Services, Windows, or both).

## 2.2.2 XSR installation validation

DB2 installation job DSNTIJRV is designed as an installation verification program for all the DB2-supplied procedures, including those used by the XSR. Run it to validate that all services are operational. An abridged version of the output of DSNTIJRV is in Example 2-22, which shows a report of the testing of the XSR procedures and the ODBC procedures.

*Example 2-22 DSNTIJRV installation verification job output*

```
DSNT040I  06.27.58  DSNTRVFY ROUTINE VALIDATION SUMMARY
    STATUS    SCHEMA            SPECIFIC NAME
 -- ------    ------            -------------
.....
 /  PASSED    SYSPROC           XSR_ADDSCHEMADOC
 /  PASSED    SYSPROC           XSR_COMPLETE
 /  PASSED    SYSPROC           XSR_REGISTER
 /  PASSED    SYSPROC           XSR_REMOVE
.....
 /  PASSED    SYSIBM            SQLCOLUMNS
 /  PASSED    SYSIBM            SQLCOLPRIVILEGES
 /  PASSED    SYSIBM            SQLFOREIGNKEYS
 /  PASSED    SYSIBM            SQLFUNCTIONCOLS
 /  PASSED    SYSIBM            SQLFUNCTIONS
 /  PASSED    SYSIBM            SQLGETTYPEINFO
 /  PASSED    SYSIBM            SQLPRIMARYKEYS
 /  PASSED    SYSIBM            SQLPROCEDURECOLS
 /  PASSED    SYSIBM            SQLPROCEDURES
 /  PASSED    SYSIBM            SQLSPECIALCOLUMNS
 /  PASSED    SYSIBM            SQLSTATISTICS
 /  PASSED    SYSIBM            SQLTABLEPRIVILEGES
 /  PASSED    SYSIBM            SQLTABLES
 /  PASSED    SYSIBM            SQLUDTS
.....

DSNT033I DSNTRVFY VALIDATION PROGRAM ENDED, RETURN CODE = 0
```

## 2.2.3 XSR setup troubleshooting

If you have difficulties using the XML schema repository, a step that might be helpful is to check the following list of potential setup issues.

- Check that both the WLM application environments for XSR are started and available, as shown in Example 2-23.

*Example 2-23 z/OS console display WLM APPLENV status*

```
D WLM,APPLENV=DSNWLMDBOB_XML

 RESPONSE=SC63
  IWM029I  14.43.07  WLM DISPLAY 072
    APPLICATION ENVIRONMENT NAME     STATE     STATE DATA
    DSNWLMDBOB_XML                   AVAILABLE
    ATTRIBUTES: PROC=DBOBWLMX SUBSYSTEM TYPE: DB2

D WLM,APPLENV=DSNWLMDBOB_JAVA

 RESPONSE=SC63
  IWM029I  14.43.39  WLM DISPLAY 074
    APPLICATION ENVIRONMENT NAME     STATE     STATE DATA
    DSNWLMDBOB_JAVA                  AVAILABLE
    ATTRIBUTES: PROC=DBOBWLMJ SUBSYSTEM TYPE: DB2
```

- Check that the XSR tables are created with the SQL statement in Example 2-24.

*Example 2-24 Check that XSR tables exist*

```
select name from sysibm.systables
   where creator = 'SYSIBM' and name like 'XSR%'

XSRANNOTATIONINFO
XSRCOMPONENT
XSROBJECTCOMPONENTS
XSROBJECTGRAMMAR
XSROBJECTHIERARCHIES
XSROBJECTPROPERTY
XSROBJECTS
XSRPROPERTY
```

- Check that the XSR schema validation routines are created with the SQL statement in Example 2-25.

*Example 2-25 Check that XSR routines exist*

```
select name from sysibm.sysroutine
   where schema= 'SYSPROC' and name like 'XSR%'

XSR_ADDSCHEMADOC
XSR_COMPLETE
XSR_REGISTER
XSR_REMOVE
```

- Check that the Java environment is correctly set up, by creating two Java user-defined functions that return the name and version number of the Java environment. Submit the DDL in Example 2-26 to create a couple of Java stored procedures.

*Example 2-26 Creation of Java-stored procedures*

```
CREATE FUNCTION SYSADM.JAVDRVV ()
   RETURNS VARCHAR(100)
   FENCED NO SQL
   LANGUAGE JAVA
   SPECIFIC JAVDRVV
   EXTERNAL NAME 'com.ibm.db2.jcc.DB2Version.getVersion'
   WLM ENVIRONMENT WLMJAVA
   NO EXTERNAL ACTION
   NO FINAL CALL
   PROGRAM TYPE SUB
   PARAMETER STYLE JAVA;

CREATE FUNCTION SYSADM.JAVDRVN ()
   RETURNS VARCHAR(100)
   FENCED NO SQL
   LANGUAGE JAVA
   SPECIFIC JAVDRVN
   EXTERNAL NAME 'com.ibm.db2.jcc.DB2Version.getDriverName'
   WLM ENVIRONMENT WLMJAVA
   NO EXTERNAL ACTION
   NO FINAL CALL
   PROGRAM TYPE SUB
   PARAMETER STYLE JAVA;
   COMMIT;

SELECT SYSADM.JAVDRVN() FROM SYSIBM.SYSDUMMY1;

-- which returns “IBM Data Server Driver for JDBC and SQLJ”

SELECT SYSADM.JAVDRVV() FROM SYSIBM.SYSDUMMY1;

-- which returns 4.11.75 on the system used in this redbook.
```

If all the checks are successful, and you still have errors using the XSR routines, you must then look at the specific error codes and address them individually. Potential errors can include `SQLCODE -805 package not found` errors if you have not run the bind jobs, or Java errors if the Java environment is not defined correctly in the `JAVASP.JSPENV` file.

We did experience problems using XSR_COMPLETE that we resolved by adding a non-APF-authorized data set to the STEPLIB concatenation in the WLM procedure that is used by XSR_COMPLETE.

### 2.2.4 z/OS XML system services

DB2 10 pureXML uses the z/OS XML system services for XML schema validation and XML parsing. These services are 100% eligible to be executed on a zIIP or zAAP processor. Figure 2-3 on page 43 shows the z/OS XML system services processing flow. If the zAAP on zIIP feature is activated zAAP-eligible z/OS XML system services, workloads are eligible to be processed on a zIIP processor.

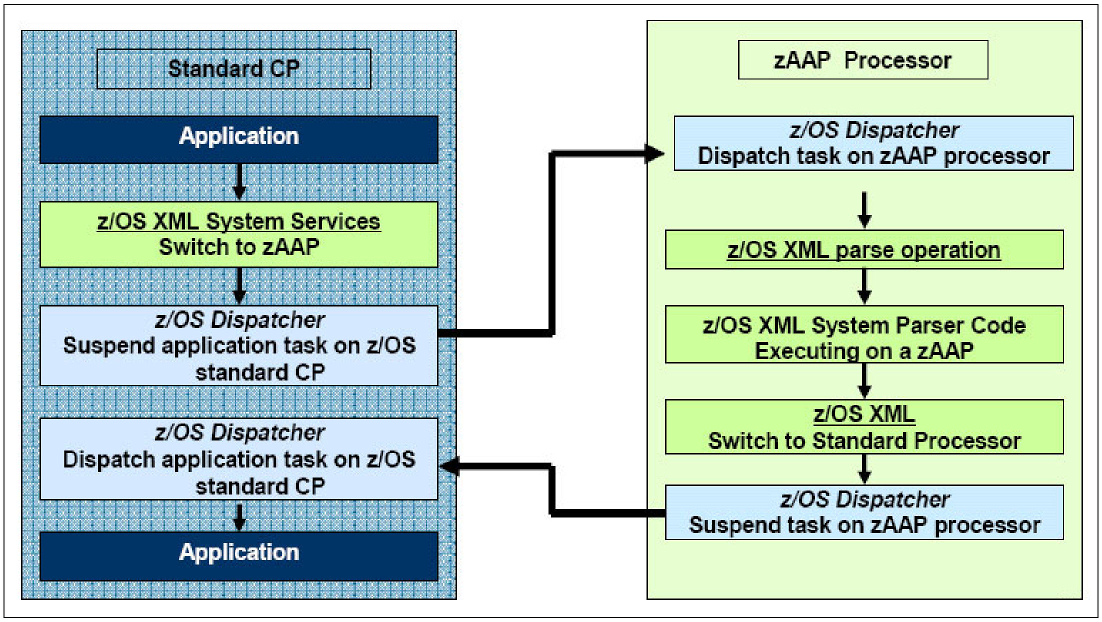

Standard CP
Application
z/OS XML System Services
Switch to zAAP
z/OS Dispatcher
Suspend application task on z/OS standard CP
z/OS Dispatcher
Dispatch application task on z/OS standard CP
Application
zAAP Processor
z/OS Dispatcher
Dispatch task on zAAP processor
z/OS XML parse operation
z/OS XML System Parser Code
Executing on a zAAP
z/OS XML
Switch to Standard Processor
z/OS Dispatcher
Suspend task on zAAP processor

*Figure 2-3 z/OS XML system services and zAAP processing flow*

## DB2 for z/OS pureXML XML parsing

DB2 pureXML invokes the z/OS XML system services for XML parsing. As a result, the XML parsing request becomes 100% zIIP- or zAAP-eligible, depending on whether the parsing or schema validation request is driven by DRDA® through a database access thread (DBAT) or through an allied DB2 thread.

## Built-in function DSN_XMLVALIDATE

In DB2 10, the SYSIBM.DSN_XMLVALIDATE function is provided inside the DB2 engine as a built-in function and uses z/OS XML System Services for XML validation. Therefore, DSN_XMLVALIDATE invocations are 100% zIIP- or zAAP-eligible in DB2 10.

DB2 9 provided XML schema validation through the SYSFUN.DSN_XMLVALIDATE external UDF. The DB2 9 DSN_XMLVALIDATE UDF was executed in task control block (TCB) mode and did not use the z/OS XML system service for XML validation. Therefore, DSN_XMLVALIDATE invocations were neither zIIP nor zAAP eligible.

APARs PK90032 and PK90040 have enabled SYSIBM.DSN_XMLVALIDATE in DB2 9 and made that activity eligible for specialty engines.

For details about migrating to the new functions, see the following web address:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db2z10.doc.xml/db2z_udfdsnxmlvalidatetobifdsnxmlvalidate.htm

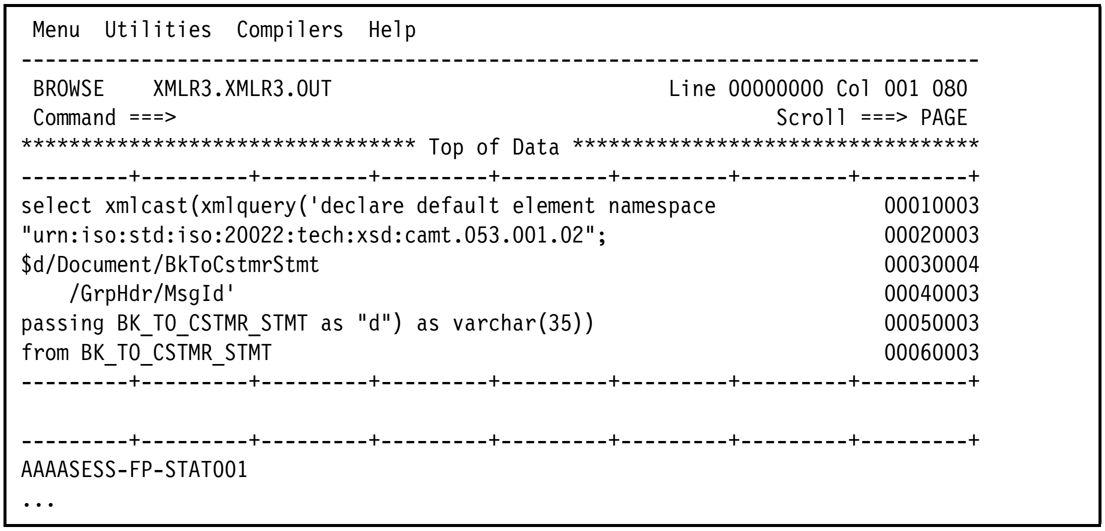
## 2.3 Choice of tools

We recognize that experienced DB2 professionals will have their own preferences for a user interface, that depends on the roles that the professionals are trying to perform, the tasks that they are trying to execute, and their personal experiences. There are plenty of tool choices on 3270 and on the GUI[2].

### 2.3.1 3270 based tools

ISPF PDF, the DB2I panel, and SPUFI work well against DB2 tables with XML columns. If you are experienced in these tools and the ISPF editor, you are able to perform all the DBA and development tasks you need.

You might frequently encounter the situation in which your SQL and DDL statements contain an XML expression that is greater than 80 bytes long. In such circumstances, you can split an XPath expression over multiple lines and it will still run successfully, as shown in Figure 2-4.



```
 Menu  Utilities  Compilers  Help
--------------------------------------------------------------------------------
 BROWSE    XMLR3.XMLR3.OUT                            Line 00000000 Col 001 080
 Command ===>                                                  Scroll ===> PAGE
********************************* Top of Data **********************************
---------+---------+---------+---------+---------+---------+---------+---------+
select xmlcast(xmlquery('declare default element namespace              00010003
"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";                       00020003
$d/Document/BkToCstmrStmt                                               00030004
    /GrpHdr/MsgId'                                                      00040003
passing BK_TO_CSTMR_STMT as "d") as varchar(35))                        00050003
from BK_TO_CSTMR_STMT                                                   00060003
---------+---------+---------+---------+---------+---------+---------+---------+

---------+---------+---------+---------+---------+---------+---------+---------+
AAAASESS-FP-STAT001
...
```

*Figure 2-4 Splitting an XPath expression over multiple lines in 3270 SPUFI session*

DB2 Administration Tool for z/OS provides database administration facilities for DB2 objects containing XML columns. XML columns are just another data type that DB2 Administration Tool caters for.

[2] GUI stands for graphical user interface.

## 2.3.2 GUI-based tools

IBM offers a range of graphical-based tools, mostly based on the Eclipse framework, which support database administration and development activities with DB2 and pureXML.

Table 2-2 summarizes several tools that are most likely to be used with DB2 and pureXML.

*Table 2-2 IBM tools for DB2 administration and development with DB2 pureXML.*

| Tool | Overview of capabilities | Comments |
|---|---|---|
| Data Studio IDE | Data development (SQL, SQL/XML, stored procedures, data web services and more)<br>XML: XML/Schema editor, XML mapper, web services, Schema registration and more | This download is included in the DB2 license for all platforms. It provides an Eclipse-based environment for a wide range of DB2 development and administration activities, including XML. |
| Optim™ Development Studio | This tool is based on Data Studio, and includes extensions, such as:<br>▶ pureQuery support<br>▶ XML validator<br>▶ XSD validator | This chargeable tool includes additional facilities beyond Data Studio, which are aimed more at a development user than a DBA. |
| InfoSphere™ Data Architect | Logical and physical data modeling to design databases, discover, relate, and integrate and standardize diverse data assets.<br>XML: Data Modeling, XML schema transformations | This chargeable tool is aimed at data modelers and architects. This tool provides logical and physical data modelling and schema development capabilities. |
| Rational® Developer for System z | Provides System z developers with tools for building traditional and composite applications in an SOA and Web 2.0 environment | This chargeable tool is aimed at System z application developers.<br>The tool contains an overlapping of XML-related tools with the products, and extended XML mapping and integration tools for System z applications. |

A comprehensive paper, “Tools and XML functionality for DB2 pureXML users” by Bryan Patterson, is available on IBM developerworks website. The paper lists a wider range of IBM tools, and describes the tasks that they support in the context of typical development and DBA roles. Access the paper at the following address:

http://www.ibm.com/developerworks/data/library/techarticle/dm-1012xmltools/index.html

# 3

# Application scenario

In this chapter, we introduce the application scenario that is based on requirement to process XML messages for business purposes. The message received contains bank statement information received from a financial institution. The scenario includes the following tasks:

- Using an openly published XML standard as the basis for documents that we store and manipulate in DB2.
- Integration with other systems through WebSphere MQ to receive XML messages that are received by an organization's enterprise service bus (ESB).
- Storing and processing the XML documents in DB2, using stored procedure, COBOL, and JAVA programs that all process XML and relational data together.
- Indexing and searching the XML tables, by using standard SQL based query tools, to enable the business and audit requirements to be satisfied.

This chapter contains the following topics:

- Requirement for XML event logging and auditing
- Application scenario
- Application code samples

© Copyright IBM Corp. 2011. All rights reserved.

## 3.1 Requirement for XML event logging and auditing

Event logging, often used for auditing purposes, is one of the simplest use cases for DB2 pureXML.

Financial services companies are growing their usage of messaging and workflow systems at a phenomenal rate. Business processes are modelled and implemented, so that they may be executed as efficiently as possible. ESB technology is used to link together all the systems that must be involved to complete these business processes. Messages are sent over an organization's enterprise service bus during the automated execution of these business processes.

Although many technologies and products might be used to implement these systems, often the common glue binds them all together is XML.

An audit trail of some or all of the XML messages that are sent and received over the ESB can be valuable for many reasons:

- An audit trail of messages describing high value financial transactions might provide a valuable source of information for monitoring and dashboard systems.
- Compliance requirements for certain applications might require that a very detailed audit trail be maintained.
- Technical problem resolution for messaging and workflow systems might be assisted by being able to trace the flow of messages through the execution of a business process.

Whatever the reasons for choosing to log event data in a persistent datastore, logging the XML messages is valuable only if you are subsequently able to read and analyze the XML messages with easy-to-use query and reporting tools.

DB2 pureXML provides an excellent platform for such event-logging systems, for the following reasons:

- XML messages can be written to DB2 with minimal application development effort. No complex relational models must be developed to log XML messages for subsequent auditing processing.
- The content of XML messages that have been stored in DB2 can be subsequently searched and accessed with ease by using SQL/XML queries.
- XML data elements that are used for searching and joining of data (such as user ID, customer ID, time, and date) can be indexed so that queries can be optimized to access only those XML messages which are needed for the required task.

## 3.2 Application scenario

The application scenario (illustrated in Figure 3-1 on page 49) in this book consists of the following aspects:

- We use the Bank To Customer Statement V2 (one of the ISO 20022 Universal financial industry message schemas) as the openly published XML standard for the XML documents that we save in DB2. ISO 20022 or UNIFI is the ISO Standard for Financial Services Messaging.
- We use the DB2 MQ Listener to receive XML messages from WebSphere MQ, and log them to DB2 using a native SQL stored procedure.

- We retrieve, manipulate, and resave those XML documents in DB2 by using stored procedures, COBOL, and JAVA programs. These programming examples show how the XML data bindings work in these languages, and use the SQL/XML functions available within DB2.
- We create XML indexes for searching the XML audit tables, and show how normal SQL query tools can search these documents to enable the auditing requirements to be satisfied.

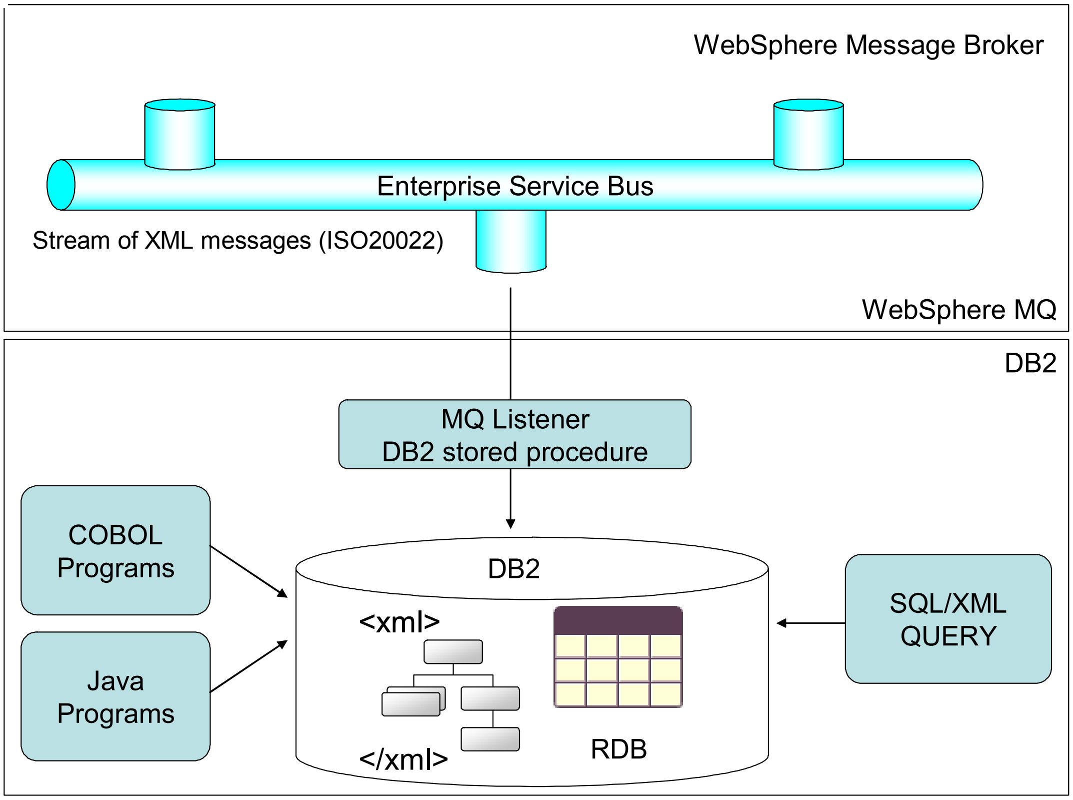


*Figure 3-1 Application scenario*

A common requirement when implementing XML data within DB2 is to work with an existing standard for XML messages. In our example, we have chosen to work with ISO 20022 (universal financial industry message scheme).

The ISO 20022 standard provides the financial industry with a common set of messages in a standardized XML syntax. See the following resources for more information:

- Current documentation for this standard:

  `http://www.iso20022.org/`

- ISO 20022 document:

  `http://www.iso20022.org/catalogue_of_unifi_messages.page`

- Description and schema of the Bank To Customer Statement V2:

  `http://www.iso20022.org/documents/general/Payments_Maintenance_2009.zip`

The programming scenarios in this book are focussed on only one message type from the ISO 20022 standard (Bank To Customer Statement V2). We chose this message type because most readers are familiar with the concept of a bank statement. For the ISO description of the message structure, download the Description of the Bank To Customer Statement V2. Briefly, the XML message structure contains the following information.

- A group header, consisting of the following items:
  - A unique message ID
  - A message creation time stamp
  - Message pagination control information
- The statement itself, consisting of the following items:
  - The statement ID (duplicate of group header msgid element)
  - The statement creation time stamp (duplicate of group header element)
  - The period that the statement covers (from date and to date)
  - The account identification details
  - One or more account balances (multiple balance types exist)
  - All the transaction entries during the period of the statement

With DB2 pureXML, it can be easy to import the XML schemas that define an XML standard into the DB2 XML schema repository, so that all XML documents stored in DB2 can be automatically (or manually) validated against different versions of the XML schema standards.

The availability of an openly available set of industry standard XML schemas is a tremendously helpful productivity boost. Chapter 5, “Validating XML data” on page 75 provides worked examples of the ISO 20022 standard schemas imported into DB2, so that the messages can be written into DB2 with full schema validation.

## 3.3 Application code samples

This book includes a number of programming samples that can be downloaded from the IBM Redbooks website and are described in Appendix B, “Additional material” on page 277. The samples are provided in three groups:

- DB2 routines (stored procedures, UDFs and so on)
- COBOL programs
- Java programs

The DDL and schema samples are common to all the programming samples. However, each programming sample is independent of the other programming samples. So, if your interest is COBOL, you can download and use the COBOL samples, without depending on Java or stored procedure samples.

Figure 3-2 on page 51 illustrates the flow between the various code samples presented in this book. The flow is explained in the following sections.

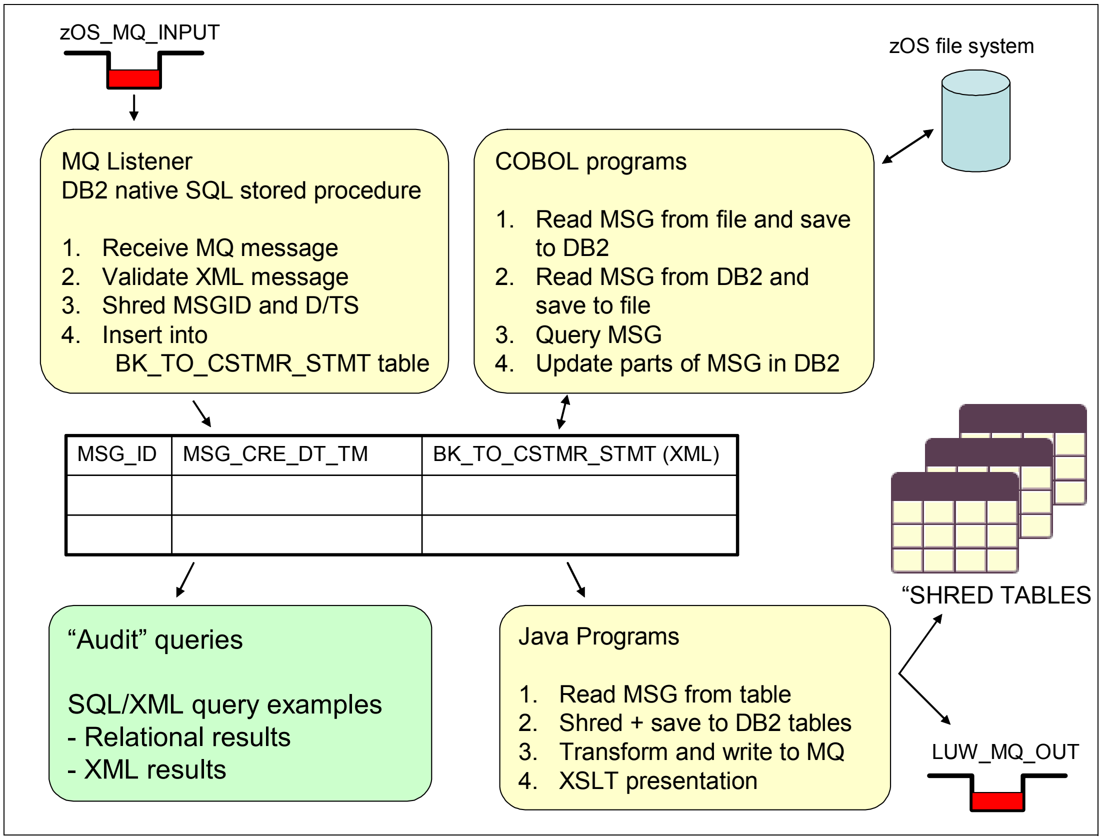


*Figure 3-2 Four application code samples*

## 3.3.1 DB2 SQL/XML programming pureXML

The amount of programming that happens within DB2 itself has been growing steadily as facilities like DB2 native SQL stored procedures have become more powerful. Stored procedures and user defined functions can be used in conjunction with external applications like COBOL and Java, where they provide reusable routines that are productive to develop and perform exceptionally well within the DB2 engine.

A number of stored procedures are illustrated in Chapter 6, "DB2 SQL/XML programming" on page 89 and cover the following tasks:

- Using XML documents as input and output parameters to stored procedures, which manipulate them (such as validating XML documents, shredding XML data elements into relational columns, transforming XML documents or sub-documents).
- Retrieving XML messages from WebSphere MQ.
- Using the DB2 MQ Listener to receive XML messages from WebSphere MQ, and automatically process them with a stored procedure.
- Receiving XML messages from change data capture products, and using them to build XML documents that contain a history of data changes.

In addition to the stored procedures, Chapter 6, "DB2 SQL/XML programming" on page 89 also contains a number of SQL/XML examples that show how the SQL/XML language can be used to query the XML documents for auditing and other purposes.

The examples in the chapter are based on programs in the `storedproceduresamples.zip` file, which is described, including download instructions, in Appendix B, “Additional material” on page 277.

## 3.3.2 Using Java with DB2 pureXML

Chapter 7, “Using XML with Java” on page 131 covers Java programming in conjunction with DB2 pureXML. The examples in this chapter are based on programs are contained in the `javasamples.zip` file, which can be downloaded as described in Appendix B, “Additional material” on page 277.

The functions of the Java programs are to perform the following tasks:

- Shred the previously saved XML message.
- Query the message to produce a new XML document.
- Output the new message to a WebSphere MQ queue.
- Use the binary XML format to retrieve XML documents from DB2 to the IBM universal driver for JDBC and SQLJ.
- Perform XSLT transformations on the XML documents.

## 3.3.3 Using COBOL with DB2 pureXML

Chapter 8, “Using XML with COBOL” on page 157 covers COBOL programming in conjunction with DB2 pureXML. The examples in the chapter are based on programs contained in the cobolsamples.zip file, which can be downloaded as described in Appendix B, “Additional material” on page 277.

Several COBOL programs are presented in Chapter 8, “Using XML with COBOL” on page 157, and they perform the following tasks:

- Read an XML message from a file and save it in DB2.
- Select an XML message from DB2 and save it to a file.
- Extract information from XML documents in DB2.
- Update an XML document on a sub-document level.

In addition, we demonstrate what impact a schema change can have on the COBOL application and describe how it compares to a relational implementation of the same database schema.

Finally, we look at some of the XML functionality available in native COBOL as a complement to the pureXML capabilities in DB2.

# 4

# Creating and adding XML data

In this chapter, we describe how to work with XML data:

- We show how to create a table with an XML column and the storage structures that are used to handle the XML column.
- We describe the support for multiple version functions that are available with DB2 10.
- We show how to retrieve information from the DB2 catalog tables for the base objects and the related XML objects, and demonstrate how an SQL INSERT statement can be used to move an XML document into an XML column.
- We include a brief overview of creating user indexes on an XML column.

This chapter contains the following topics:

- Creating and adding XML data
- Storage structure for XML data
- Multiple version concurrency control for XML
- Catalog queries to gather information
- Display database command
- Ingesting XML data
- XML indexes

© Copyright IBM Corp. 2011. All rights reserved.

## 4.1 Creation of tables with XML columns

To create tables with XML columns, you specify columns with the XML data type in the CREATE TABLE statement. A table can have one or more XML columns.

You do not specify a length when you define an XML column. There is no architectural limit on the size of an XML value in a database. However, textual XML data that is exchanged with a DB2 database is limited to 2 GB minus 1, so the effective limit of an XML column is 2 GB-1.

As with a LOB column, an XML column holds only a descriptor of the column. The data is stored separately.

Example 4-1 shows how you can define the BK_TO_CSTMR_STMT table, referred to in 3.3, "Application code samples" on page 50 for the application scenario.

*Example 4-1 BK_TO_CSTMR_STMT table with XML column*

```
CREATE TABLE BK_TO_CSTMR_STMT
   (MSG_ID              VARCHAR(35),
    MSG_CRE_DT_TM       TIMESTAMP WITH TIMEZONE,
    BK_TO_CSTMR_STMT    XML NOT NULL)
```

Because the IN clause is not specified, the table is created in an implicitly created partition-by-growth universal table space with a value of 256 for MAXPARTITIONS. The additional table space for the XML column is created by using the default storage group SYSDEFLT in an implicitly created database.

## 4.2 Storage structure for XML data

The storage structure for XML data is similar to the storage structure for LOB data.

As with LOB data, the table that contains an XML column (the base table) is in a different table space from the table space which contains the XML data.

The storage structure depends on the type of table space that contains the base table.

We show the relationship between non-partitioned table space for base tables with XML columns and the corresponding XML table spaces and tables.

A base table space can be *segmented* as shown in Figure 4-1.

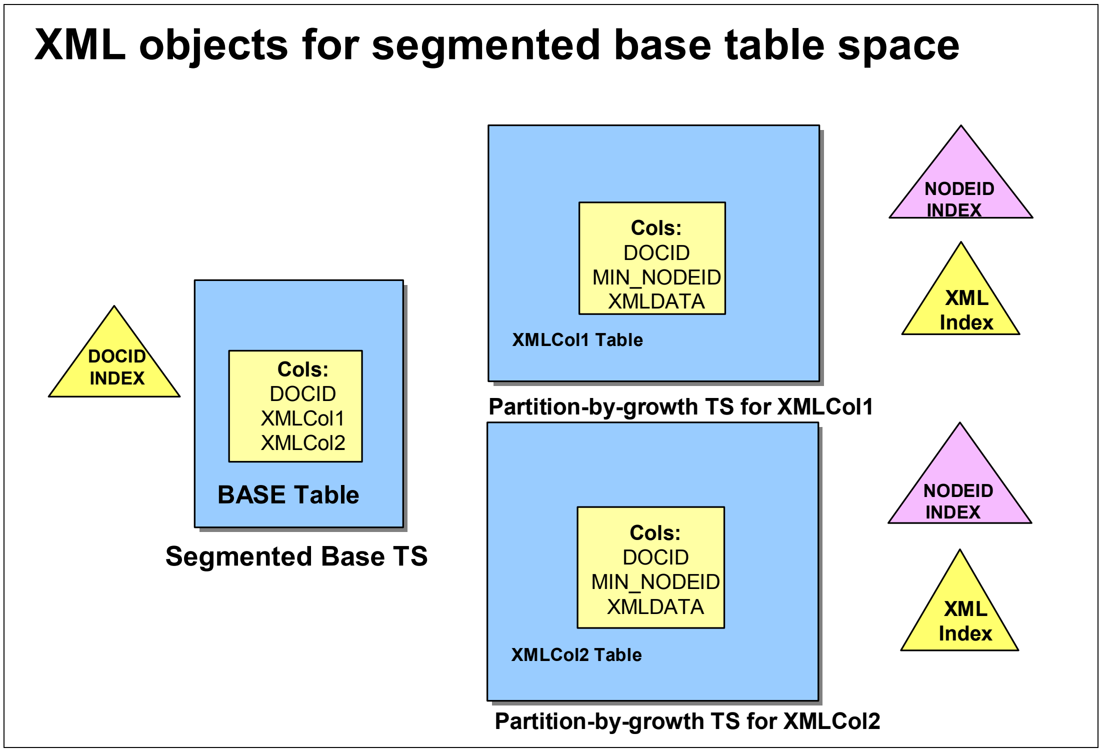


*Figure 4-1 XML objects for segmented base table space*

A base table space can be *partitioned-by-growth* as shown in Figure 4-2.

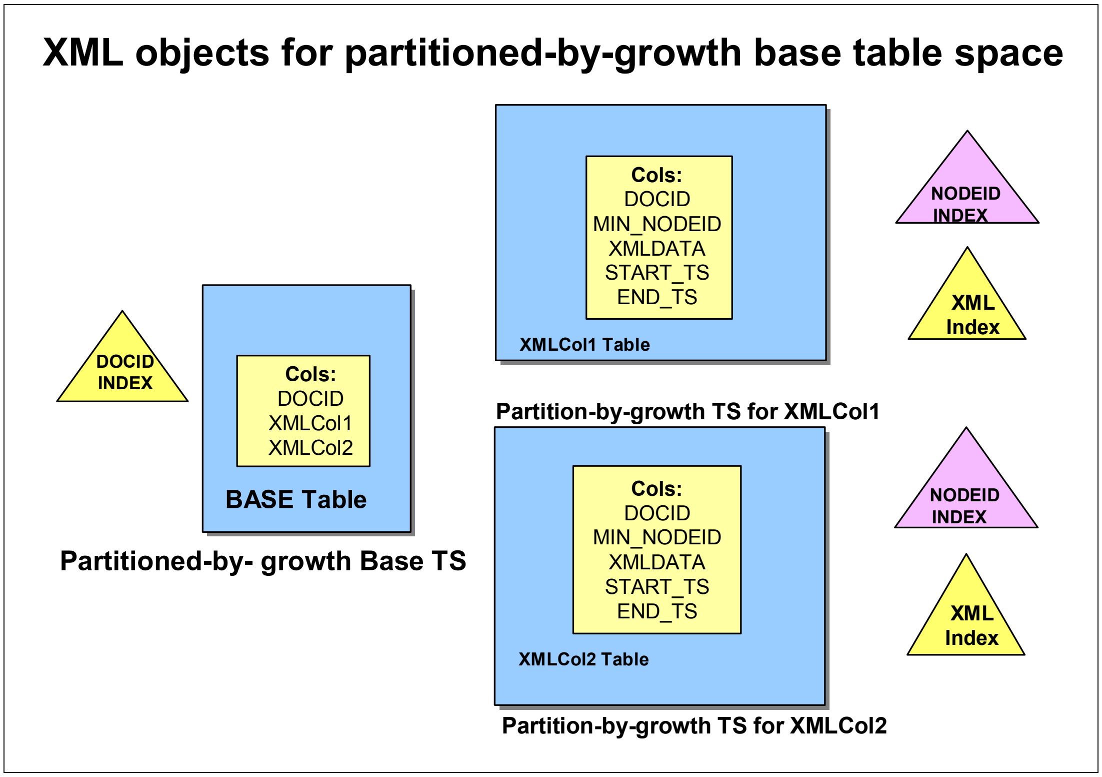


*Figure 4-2 XML objects for partition-by-growth base table space*

In both cases the XML table space is partition-by-growth. When you issue the CREATE TABLE statement which includes XML columns for a non-partitioned table or the ALTER TABLE statement to add an XML column to a non-partitioned table, the following objects are created implicitly by DB2 to support the XML columns:

- A column named DB2_GENERATED_DOCID_FOR_XML that is defined as NOT NULL

  We refer to this column as DOCID column. DOCID uniquely represents each row. This column is hidden. For each table, DB2 needs only one DOCID column even if you add more columns with data type XML. This DOCID is defined as GENERATED ALWAYS, meaning that you cannot update the DOCID column.

- An XML indicator column in the base table for each XML column

  The XML indicator column is treated as a varying length column. An XML indicator column is VARCHAR(6) if the base table is segmented. It is stored in the base table in place of an XML column, and indicates whether the XML value for the column is null or zero length. The XML indicator column is VARCHAR(14) if the base table is in a partition-by growth universal table space. The extra eight bytes are used to support multiple versions of XML documents.

- A unique index on the DOCID column

  This index is known as a DOCID index. The DOCID index key is only the document ID itself that points to the base table RID.

- An XML table space (partition-by-growth table space) per XML column which uses the Unicode UTF-8 encoding scheme

- An XML table with columns DOCID BIGINT, MIN_NODEID VARBINARY(128), and XMLDATA VARBINARY(15850) if the base table space is segmented

  The XML table has two more columns START_TS and END_TS (used to support multiple versions of XML documents) if the base table space is partition-by-growth.

- An index on columns DOCID and XMLDATA in each XML table that DB2 uses to maintain document order, and map logical node IDs to physical record IDs

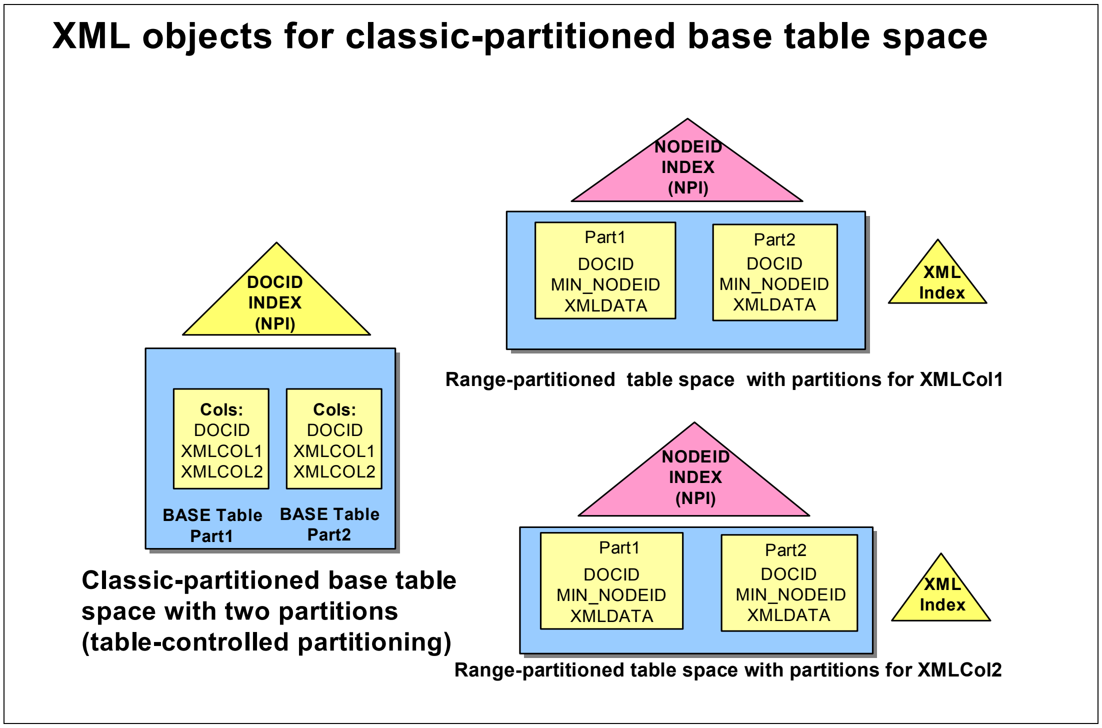
  This index is known as a NODEID index. The NODEID index is an extended, non partitioning index.

- DOCID and MIN_NODEID are used for row (XMLDATA) clustering and sort records in document order so prefetch will work.

- An XML document can span more than one partition. The base table space and the XML table space grow independently.

We show the relationship between partitioned table space for base tables with XML columns and the corresponding XML table spaces and tables.

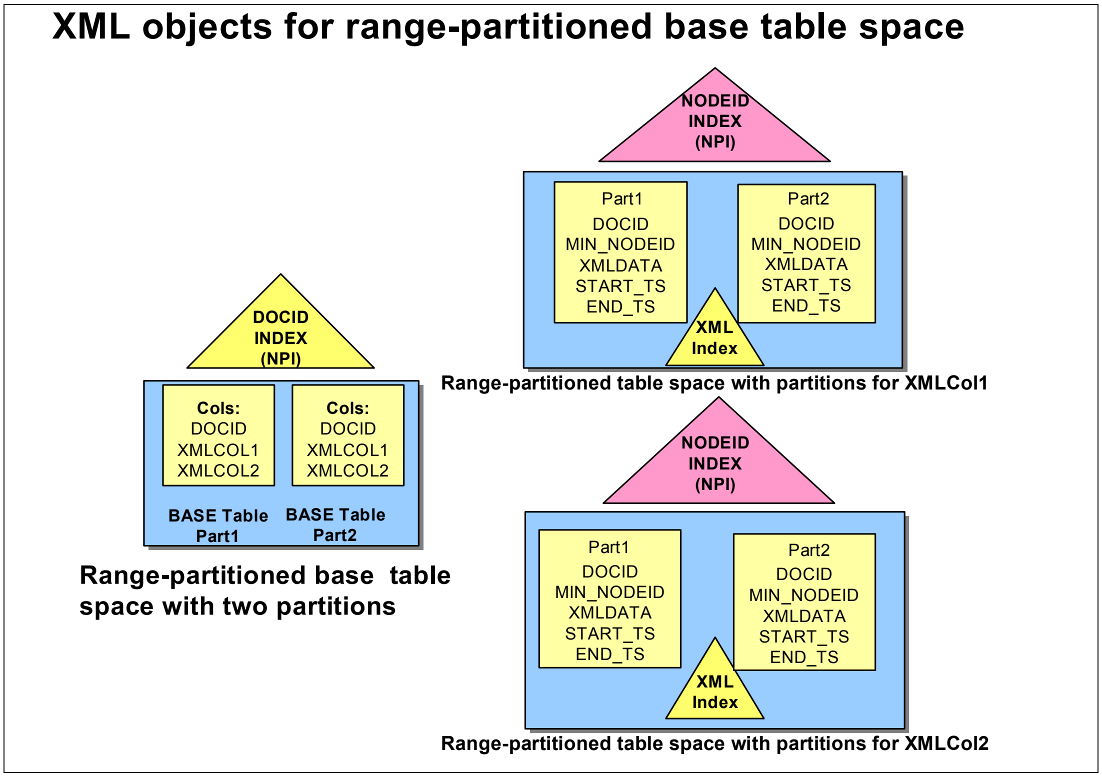
A partitioned base table space can be *classic-partitioned* as shown in Figure 4-3.



XML objects for classic-partitioned base table space
NODEID INDEX (NPI)
Part1 DOCID MIN_NODEID XMLDATA
Part2 DOCID MIN_NODEID XMLDATA
XML Index
Range-partitioned table space with partitions for XMLCol1
DOCID INDEX (NPI)
Cols: DOCID XMLCOL1 XMLCOL2
Cols: DOCID XMLCOL1 XMLCOL2
BASE Table Part1
BASE Table Part2
Classic-partitioned base table space with two partitions (table-controlled partitioning)
NODEID INDEX (NPI)
Part1 DOCID MIN_NODEID XMLDATA
Part2 DOCID MIN_NODEID XMLDATA
XML Index
Range-partitioned table space with partitions for XMLCol2

*Figure 4-3 XML objects for classic-partitioned base table space*

A partitioned table space can be *range-partitioned* shown in Figure 4-4.



XML objects for range-partitioned base table space
NODEID INDEX (NPI)
Part1 DOCID MIN_NODEID XMLDATA START_TS END_TS
Part2 DOCID MIN_NODEID XMLDATA START_TS END_TS
XML Index
Range-partitioned table space with partitions for XMLCol1
DOCID INDEX (NPI)
Cols: DOCID XMLCOL1 XMLCOL2
Cols: DOCID XMLCOL1 XMLCOL2
BASE Table Part1
BASE Table Part2
Range-partitioned base table space with two partitions
NODEID INDEX (NPI)
Part1 DOCID MIN_NODEID XMLDATA START_TS END_TS
Part2 DOCID MIN_NODEID XMLDATA START_TS END_TS
XML Index
Range-partitioned table space with partitions for XMLCol2

*Figure 4-4 XML objects for range-partitioned base table space*

When you issue the CREATE TABLE statement that includes XML columns for a partitioned table or the ALTER TABLE statement to add an XML column to a partitioned table, the following objects are created implicitly by DB2 to support the XML columns:

- A column called DB2_GENERATED_DOCID_FOR_XML that is defined as NOT NULL

  We refer to this column as a DOCID column. DOCID uniquely represents each row. This column is hidden. For each table, DB2 needs only one DOCID column even if you add more columns with data type XML. This DOCID is defined as generated always, meaning that you cannot update the DOCID column.

- A unique index on the DOCID column

  This index is known as a DOCID index. The DOCID index key is simply the document ID itself pointing to the base table RID. The DOCID index is a non-partitioning index.

- An XML indicator column in the base table for each XML column

  The XML indicator column is treated like a varying length column. An XML indicator column is VARCHAR(6). It is stored in the base table in place of an XML column, and indicates whether the XML value for the column is null or zero length. The XML indicator column is VARCHAR(14) if the base table is in a range partitioned universal table space. The extra eight bytes are used to support multiple versions of XML documents.

- An XML table space (range-partitioned table space) per XML column, which uses the Unicode UTF-8 encoding scheme

  Each XML table space contains one table with the corresponding number of parts. The XML table is partitioned only on the basis of the base row partition. Although it is partitioned, the XML table space does not have limit keys. The XML data resides in the partition number that corresponds to the partition number of the base row. If a row changes partition in the base table, the XML document moves as well.

- An XML table with columns DOCID BIGINT, MIN_NODEID VARBINARY(128), and XMLDATA VARBINARY(15850) if the base table space is classic partitioned

  The XML table has two more columns START_TS and END_TS (used to support multiple versions of XML documents) if the base table space is range-partitioned.

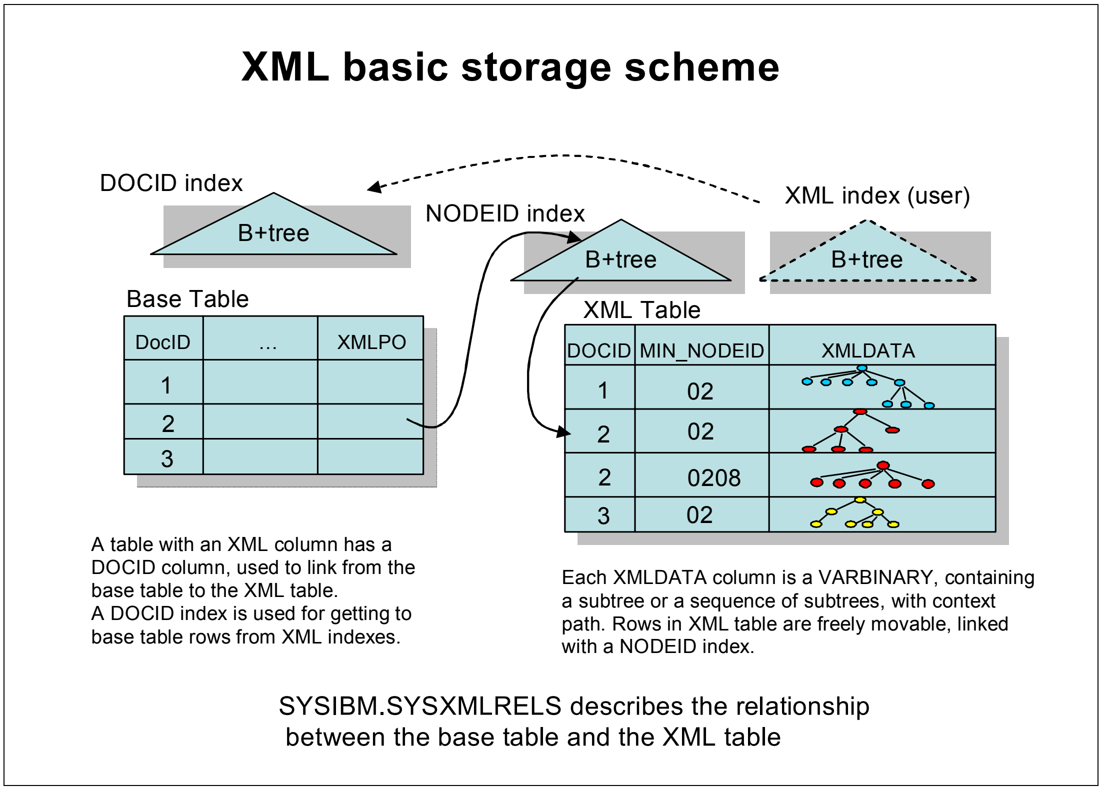
- An index on columns DOCID and XMLDATA in each XML table that DB2 uses to maintain document order, and map logical node IDs to physical record IDs

  This index is known as a NODEID index. The NODEID index is an extended, non-partitioning index.

- DOCID and MIN_NODEID are used for row (XMLDATA) clustering and sort records in document order so prefetch will work.

**Important:** The implicitly created XML table space is *always* a universal table space, either partition-by-growth or range-partitioned. If you want to have the ability to handle multiple versions of XML document make sure the base table space is a universal table space.

Figure 4-5 on page 59 shows the storage structure of XML data when XML versions are not defined.



| DocID | … | XMLPO |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

| DOCID | MIN_NODEID | XMLDATA |
|---|---|---|
| 1 | 02 | |
| 2 | 02 | |
| 2 | 0208 | |
| 3 | 02 | |

A table with an XML column has a DOCID column, used to link from the base table to the XML table.
A DOCID index is used for getting to base table rows from XML indexes.

Each XMLDATA column is a VARBINARY, containing a subtree or a sequence of subtrees, with context path. Rows in XML table are freely movable, linked with a NODEID index.

SYSIBM.SYSXMLRELS describes the relationship between the base table and the XML table

*Figure 4-5 XML basic storage scheme*

Note the following information:

- The XML column tells DB2 which NODEID index to search.
- NODEID index is made up of the DOCID and a NODEID.
- Key of DOCID+NODEID in the index tells DB2 which RID to get from XML tables.
- NODEID Index may contain multiple entries per XML data row depending on the nodes grouped in the XML data row.
- The NODEID index is created implicitly on the XML table for each XML column that is added to the base table. The XML node id uniquely identifies a node within a document.
- To locate the corresponding XML column, DB2 uses the document ID from the base row and pairs that with an XML node ID in the XML table, starting with the minimum node ID, to search the NODEID index for the XML data.
- The catalog table SYSIBM.SYSXMLRELS contains one row for each XML table that is created for an XML column to provide information about the relationship between XML column and XML table.

The node ID for a node is the concatenation of local node ids contained in each node along the path from the root to the node. For each XML data record, a context node ID exists that contains node IDs from the root to the parent node for the nodes inside the record. The MIN_NODEID column of an XML table contains the minimum node ID within the XMLDATA record in that row, which is a concatenation of the context node ID and the first node ID in the record body. DOCID and MIN_NODEID are used for clustering rows that belong to the same document.

**XML indexes:** XML indexes are user-created indexes for achieving good performance.

## 4.3 Multiple version concurrency control for XML

DB2 supports multiple versions of an XML document in an XML column if the *base table space* for the table that contains the XML column is also a *universal table space* and created in DB2 10 NFM. All XML columns in the table support multiple versions.

> **Support:** If the base table space is not a universal table space, it does not support multiple XML versions. To convert the base table space from either segmented or partitioned to universal table space, you must remove it (with the DROP command) and re-create it. ALTER and REORG are not sufficient in this case.

With XML versions, when you insert an XML document into an XML column, DB2 assigns a version number to the XML document. If the entire XML document is updated, DB2 creates a new version of the document in the XML table. If a portion of the XML document is updated, DB2 creates a new version of the updated portion. When DB2 uses XML versions, more data set space is required than when versions are not used. However, DB2 periodically deletes versions that are no longer needed. In addition, you can run the REORG utility against the XML table space that contains the XML document to remove unneeded versions. DB2 removes versions of a document when update operations that require the versions are committed, and when no readers reference the unneeded versions.

> **XML versions:** XML versions differ from table space versions or index versions. The purpose of XML versions is to optimize concurrency and memory usage. The purpose of table space and index versions is to maximize data availability.

### 4.3.1 Example of improved concurrency with XML versions

This example demonstrates how multiple XML versions can improve concurrency when the same XML documents are modified multiple times within the same transaction.

Suppose that table T1, which is in a universal table space, is defined as follows:

```
CREATE TABLE T1(
  INT1 INT,
  XML1 XML,
  XML2 XML );
```

Table 4-1 shows the data in table T1.

*Table 4-1 Data in table T1*

| INT1 | XML1 | XML2 |
|---|---|---|
| 350 | <A1>111</A1> | <A2>aaa</A2> |
| 100 | <A1>111</A1> | <A2>aaa</A2> |
| 250 | <A1>111</A1> | <A2>aaa</A2> |

An application performs SQL read operations that are represented by the following pseudocode:

```
EXEC SQL
 DECLARE CURSOR C1 FOR
 SELECT INT1, XML1
```

```
  FROM T1
  ORDER BY INT1
  FOR READ ONLY;
```

At the same time, another application performs SQL write operations that are represented by the following pseudocode:

```
EXEC SQL UPDATE T1
 SET XML1 = XMLPARSE(DOCUMENT '<B1>222</B1>');
EXEC SQL OPEN CURSOR C1; (Note: Cursor C1 is in another application as described)
EXEC SQL UPDATE T1
 SET XML1 = XMLPARSE(DOCUMENT '<C1>333</C1>');
EXEC SQL FETCH FROM C1 INTO :HVINT1, :HVXML1;
```

With multiple versions, the reading application does not need to hold a lock. Thus, the updating application can do its update operations without waiting for the reading application to finish. The reading application reads the old versions of the XML values, which are consistent data.

## 4.3.2 Example of improved storage usage with XML versions

This example demonstrates how multiple XML versions can result in the use of less real storage when an XML document is the object of a self-referencing update operation.

Assume the same table T1 and data rows.

An application performs SQL operations that are represented by the pseudocode in Example 4-2

*Example 4-2 Pseudocode*

```
XEC SQL
 UPDATE T1
 SET XML1 = XML2, 1
  XML2 = XML1 2
 WHERE INT1 = 100;
EXEC SQL
 COMMIT 3 ;
```

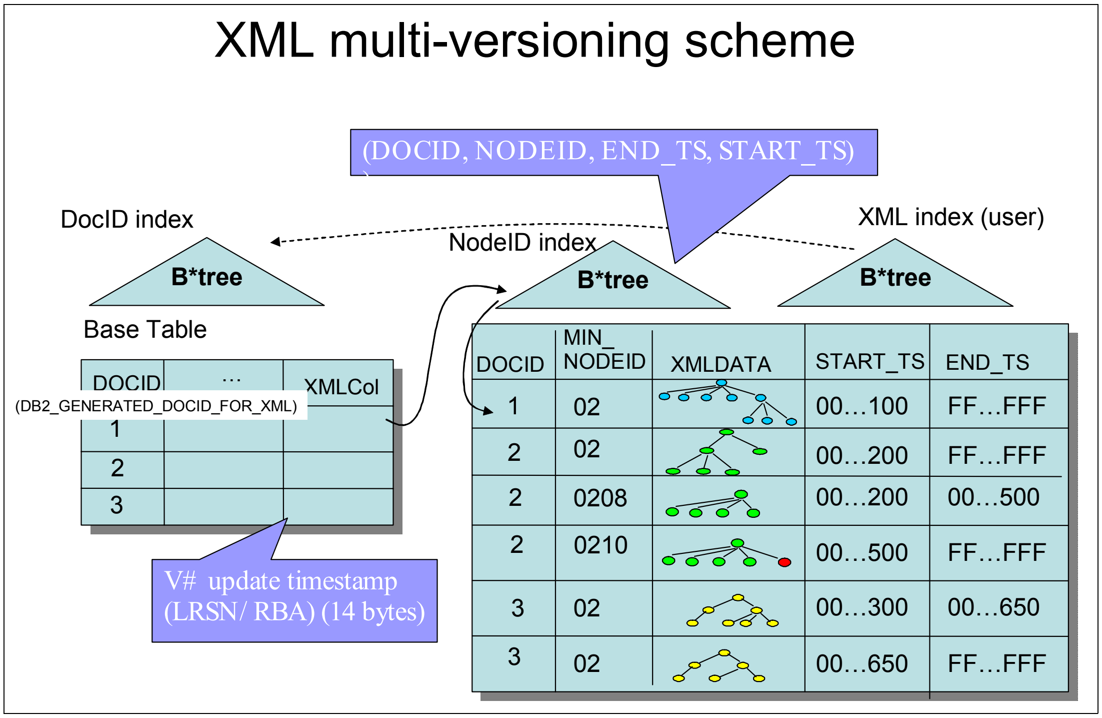
The results of those operations are as follows:

1. When column XML1 is updated, DB2 stores the updated document as a new version in the XML table for column XML1. There are now two versions of the XML document for the second row of column XML1:
   - First version: <A1>111</A1>
   - Second version: <A2>aaa</A2>
2. When column XML2 is updated, DB2 stores the updated document as a new version in the XML table for column XML2. There are now two versions of each XML document for the second row of column XML2:
   - First version: <A2>aaa</A2>
   - Second version: <A1>111</A1>
3. The update operations are committed. Thus, the old versions are no longer needed. DB2 deletes those versions from the XML tables for columns XML1 and XML2. (assuming no other readers are interested in reading these values).

Without multiple XML versions, DB2 must copy the original versions of the updated documents into memory, so that their values are not lost. For large XML documents, storage shortages might result.

### 4.3.3 Storage structure for XML data with versions

Figure 4-6 shows the storage structure of XML data to support XML versions.


XML multi-versioning scheme
(DOCID, NODEID, END_TS, START_TS)
DocID index
B*tree
NodeID index
B*tree
XML index (user)
B*tree
Base Table
V# update timestamp (LRSN/ RBA) (14 bytes)

| DOCID (DB2_GENERATED_DOCID_FOR_XML) | … | XMLCol |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

| DOCID | MIN_ NODEID | XMLDATA | START_TS | END_TS |
|---|---|---|---|---|
| 1 | 02 | | 00…100 | FF…FFF |
| 2 | 02 | | 00…200 | FF…FFF |
| 2 | 0208 | | 00…200 | 00…500 |
| 2 | 0210 | | 00…500 | FF…FFF |
| 3 | 02 | | 00…300 | 00…650 |
| 3 | 02 | | 00…650 | FF…FFF |

*Figure 4-6 XML multiple version scheme*

When you create a table with XML columns or alter a table to add XML columns, DB2 implicitly creates the following objects:

- A table space and table for each XML column

  The data for an XML column is stored in the corresponding table.

  DB2 creates the XML table space and table in the same database as the table that contains the XML column (the *base table*). The XML table space is in the Unicode UTF-8 encoding scheme.

  If the base table contains XML columns that support XML versions, each XML table contains two more columns than an XML table for an XML column that does not support XML versions. Those columns are named START_TS and END_TS, and they have the BINARY(8) data type. START_TS contains the RBA or LRSN of the logical creation of an XML record. END_TS contains the RBA or LRSN of the logical deletion of an XML record. START_TS and END_TS identify the rows in the XML table that make up a version of an XML document.

  Column START_TS represents the time when that row is created, and column END_TS represents the time when the row is deleted or updated. Column END_TS contains `X'FFFFFFFFFFFFFFFF'` initially. To avoid compression causing update overflow, columns up to column END_TS are not compressed in the reordered row format.

- If the base table space supports XML versions, the length of the XML indicator column is eight bytes longer that the XML indicator column in a base table space that does not support XML versions, that is, 14 bytes instead of six bytes.
- A document ID column in the base table, named DB2_GENERATED_DOCID_FOR_XML, with data type BIGINT

  We refer to this object as a DOCID column. The DOCID column holds a unique document identifier for the XML columns in a row. One DOCID column is used for all XML columns.

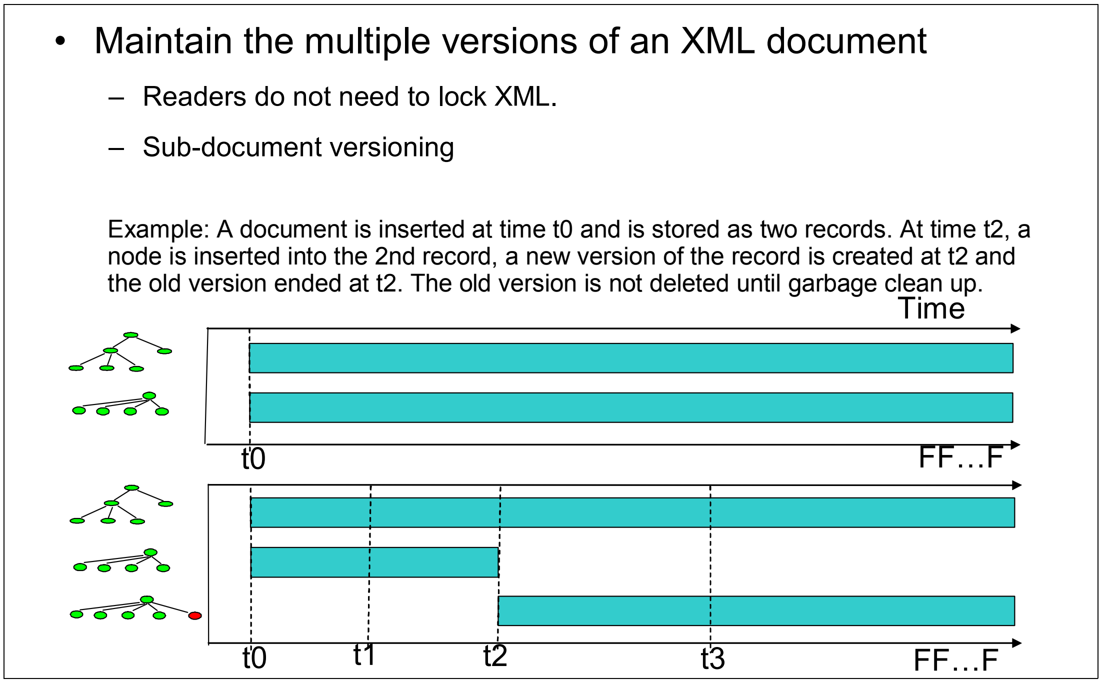
  The DOCID column has the GENERATED ALWAYS attribute. Therefore, a value in this column cannot be NULL. However, It can be null for rows that existed before a table is altered to add an XML column.
- An index on the DOCID column

  This index is known as a document ID (or DOCID) index.
- An index on each XML table that DB2 uses to maintain document order, and map logical node ids to physical record IDs

  This index is known as a NODEID index. The NODEID index is an extended, non-partitioning index.

  If the base table space supports XML versions, the index key for the NODEID index contains two more columns than the index key for a node id index for a base table space that does not support XML versions. These are START_TS and END_TS.

Figure 4-7 shows, in general, how DB2 handles support of multiple versions for XML data.



- Maintain the multiple versions of an XML document
  - Readers do not need to lock XML.
  - Sub-document versioning

Example: A document is inserted at time t0 and is stored as two records. At time t2, a node is inserted into the 2nd record, a new version of the record is created at t2 and the old version ended at t2. The old version is not deleted until garbage clean up.

*Figure 4-7 Multiple versions for XML data*

Each row in the XML auxiliary table is associated with two temporal attributes (start and end) to represent the period when the row exists:

- Start represents the time when that row is created.
- End represents the time when the row is deleted or expired.

For example, an XML document is stored as two rows in the XML auxiliary table at time t0, the second row is modified at t2,DB2 set that row as *expired* at t2, create a new row representing the modified version with create time t2. The first row is not changed during this process.

A row in XML auxiliary table is never deleted until the garbage collector cleans it up.

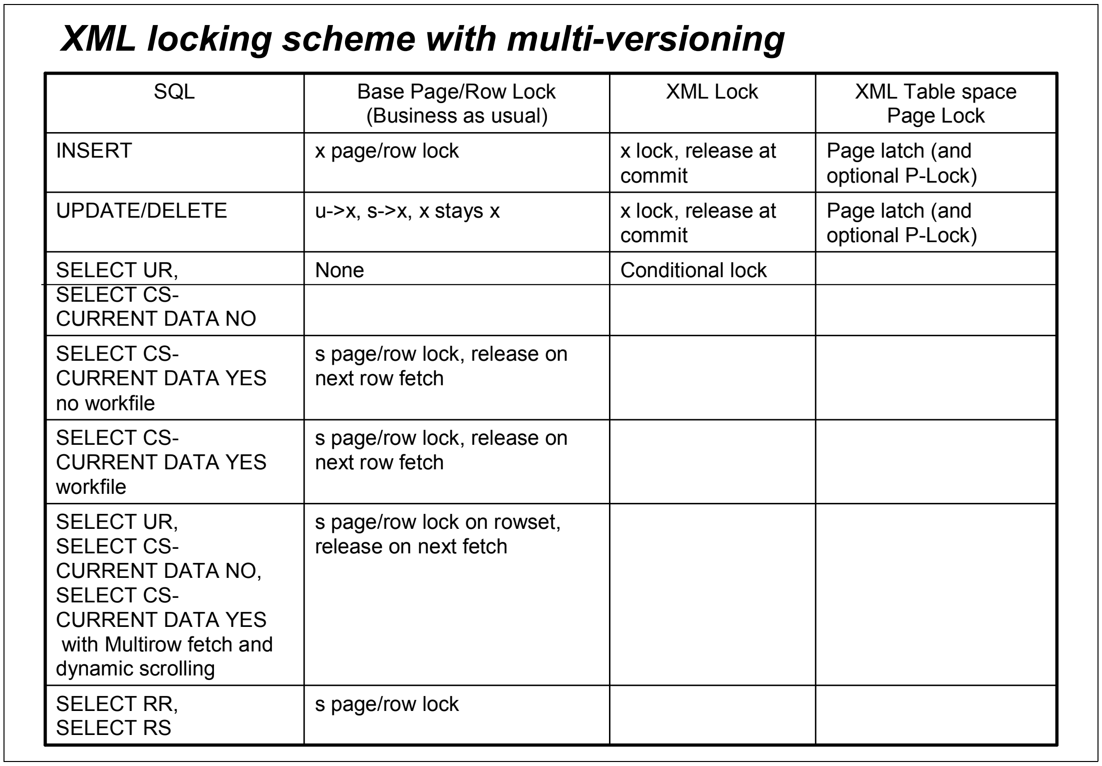
When an XML document is deleted at time t2, all the records for that document are marked *expired* at t2. When a row of an XML document is updated, all the records for that document are marked *expired* at t2, the new document is inserted into XML auxiliary table with start time set to t2.

When a part of an XML document is modified, only the existing record (or records) to be modified expire and a new version of those records is created.

**Storage structure:** This storage structure is possible only in new-function mode and for universal table spaces. Storage structure for multiversioning is a prerequisite for several other XML enhancements such as the following enhancements:

- Access of currently committed data
- "As Of" option for time-oriented data
- XML update with XMLMODIFY
- Removing restrictions for SELECT FROM UPDATE//DELETE for XML

Figure 4-8 shows the XML locking scheme in DB2 10.



**XML locking scheme with multi-versioning**

| SQL | Base Page/Row Lock (Business as usual) | XML Lock | XML Table space Page Lock |
|---|---|---|---|
| INSERT | x page/row lock | x lock, release at commit | Page latch (and optional P-Lock) |
| UPDATE/DELETE | u->x, s->x, x stays x | x lock, release at commit | Page latch (and optional P-Lock) |
| SELECT UR, | None | Conditional lock | |
| SELECT CS-CURRENT DATA NO | | | |
| SELECT CS-CURRENT DATA YES no workfile | s page/row lock, release on next row fetch | | |
| SELECT CS-CURRENT DATA YES workfile | s page/row lock, release on next row fetch | | |
| SELECT UR,<br>SELECT CS-CURRENT DATA NO,<br>SELECT CS-CURRENT DATA YES with Multirow fetch and dynamic scrolling | s page/row lock on rowset, release on next fetch | | |
| SELECT RR,<br>SELECT RS | s page/row lock | | |

*Figure 4-8 XML locking scheme with multiversioning*

## 4.4 Catalog queries to gather information

Example 4-3 describes the following tasks:

- You can obtain the implicitly created database and table space names for the BK_TO_CSTMR_STMT table from the SYSIBM.SYSTABLES catalog table as shown in **Query 1**.
- After you know the name of the implicitly created database, you can retrieve information about the default storage group, default buffer pool used for data, and default buffer pool used for indexes from the SYSIBM.SYSDATABASE catalog table as shown in **Query 2**. The Y value in the IMPLICIT column indicates the database is implicitly created.
- After you know the name of the implicitly created database, you can retrieve information about table spaces in this database from the SYSIBM.SYSTABLESPACE catalog table as shown in **Query 3**.

  There are two rows in this table:

  – The first row is for the base table space BKRTORCS. The G value in the TYPE column indicates this is a partition-by-growth universal table space, currently has one partition (indicated by the PARTITIONS column) and can grow to a maximum of 256 partitions (indicated by column MAXPARTITIONS). The Y value in the IMPLICIT column indicates the table is implicitly created. SEGSIZE used is 32. The maximum data set size is 4 GB, indicated by the DSSIZE column in kilobytes.

  – The second row is for the XML table space XBKR0000. The P value in the TYPE column indicates this is an implicit table space created for XML columns, currently has one partition (indicated by PARTITIONS column), and can grow to a maximum of 256 partitions (indicated by MAXPARTITIONS column). The Y value in the IMPLICIT column indicates the table is implicitly created. The SEGSIZE that is used is 32. The maximum data set size is 4 GB, indicated by the DSSIZE column in kilobytes.

> **Note:** The name of the XML table space always starts with X and it is always a universal table space. In this case, it is partition-by-growth because the base table space is partition-by-growth, and is created in the same database as the base table space.

*Example 4-3 Catalog queries (1 of 3)*

```
-- Query 1

SELECT SUBSTR(NAME,1,20) AS NAME,
       SUBSTR(DBNAME,1,10) AS DBNAME,
       SUBSTR(TSNAME,1,10) AS TSNAME
FROM SYSIBM.SYSTABLES
WHERE NAME='BK_TO_CSTMR_STMT' AND CREATOR=USER;
---------+---------+---------+---------+---------+---------+---
NAME                 DBNAME      TSNAME
---------+---------+---------+---------+---------+---------+---
BK_TO_CSTMR_STMT     DSN00242    BKRTORCS
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---
-- Query 2

SELECT SUBSTR(NAME,1,10) AS NAME,
       SUBSTR(CREATOR,1,10) AS CREATOR,
       SUBSTR(STGROUP,1,10) AS STGROUP,
BPOOL,INDEXBP,IMPLICIT
```

```
FROM SYSIBM.SYSDATABASE
WHERE NAME='DSN00242' ;
---------+---------+---------+---------+---------+---------+----
NAME        CREATOR     STGROUP     BPOOL     INDEXBP   IMPLICIT
---------+---------+---------+---------+---------+---------+----
DSN00242    SYSIBM      SYSDEFLT    BP0       BP0       Y
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+----
-- Query 3

SELECT SUBSTR(NAME,1,10) AS NAME,
       SUBSTR(CREATOR,1,10) AS CREATOR,
       SUBSTR(DBNAME,1,10) AS DBNAME,
       STATUS,TYPE,SEGSIZE,PARTITIONS,
       MAXPARTITIONS AS MAXP,
       DSSIZE,IMPLICIT
FROM SYSIBM.SYSTABLESPACE
WHERE DBNAME = 'DSN00242'
  AND CREATOR = USER ;
---------+---------+---------+---------+---------+---------+---------+---------+-------
NAME      CREATOR  DBNAME    STATUS  TYPE  SEGSIZE  PARTITIONS  MAXP   DSSIZE  IMPLICIT
---------+---------+---------+---------+---------+---------+---------+---------+-------
BKRTORCS  XMLR4    DSN00242  A       G          32           1   256  4194304  Y
XBKR0000  XMLR4    DSN00242  A       P          32           1   256  4194304  Y
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

Example 4-4 on page 67 describes the following tasks:

- After you know the name of the implicitly created database, you can retrieve information about tables in this database from the SYSIBM.SYSTABLES catalog table as shown in **`Query 4`**. The table has two rows:
  - The first row is for the BK_TO_CSTMTR_STMT base table (indicated by the `T` value in the TYPE column). The blank value in STATUS column indicates that the table has no unique constraint (primary key or unique key) and the definition of the table is complete. The blank value in TABLESTATUS column indicates that the table definition is complete.
  - The second row is for the XML table XBK_TO_CSTMTR_STMT. The `P` value in the TYPE column indicates this is an Implicit table created for XML columns. The blank value in the STATUS column indicates that the table has no unique constraint (primary key or unique key) and the definition of the table is complete. The blank value in the TABLESTATUS column indicates that the table definition is complete.

    **Note:** The name of the XML table always starts with X.

- You can get information about the columns in both the base table and XML table from SYSIBM.SYSCOLUMNS as shown in **`Query 5`**. The first three rows are for the three columns that are defined in the base table. The fourth row is the DOCID column that is generated by DB2. The next three rows are for columns that are defined in the XML table. The last two rows are for the two extra columns that are defined in the XML table to support multiple versions for XML documents and are present because the base table is in partition-by-growth universal table space.

*Example 4-4 Catalog queries (2 of 3)*

```
-- Query 4

SELECT SUBSTR(NAME,1,20) AS NAME,
       SUBSTR(CREATOR,1,10) AS CREATOR,
       TYPE,
       SUBSTR(DBNAME,1,10) AS DBNAME,
       SUBSTR(TSNAME,1,10) AS TSNAME,
       STATUS,TABLESTATUS
FROM SYSIBM.SYSTABLES
WHERE DBNAME = 'DSN00242'
  AND CREATOR = USER ;
---------+---------+---------+---------+---------+---------+---------+---------+
NAME                  CREATOR     TYPE  DBNAME      TSNAME   STATUS TABLESTATUS
---------+---------+---------+---------+---------+---------+---------+---------+
BK_TO_CSTMR_STMT      XMLR4       T     DSN00242    BKRTORCS
XBK_TO_CSTMR_STMT     XMLR4       P     DSN00242    XBKR0000
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+--
-- Query 5

SELECT SUBSTR(NAME,1,30) AS NAME,
       SUBSTR(TBNAME,1,20) AS TBNAME,
       SUBSTR(TBCREATOR,1,10) AS TBCREATOR,
       SUBSTR(COLTYPE,1,8) AS COLTYPE,
       LENGTH,COLNO
FROM SYSIBM.SYSCOLUMNS
WHERE TBCREATOR = USER
  AND TBNAME IN ('BK_TO_CSTMR_STMT',’XBK_TO_CSTMR_STMT’)
ORDER BY TBNAME,COLNO;
---------+---------+---------+---------+---------+---------+---------+---------+---------+
NAME                            TBNAME                TBCREATOR   COLTYPE   LENGTH   COLNO
---------+---------+---------+---------+---------+---------+---------+---------+---------+
MSG_ID                          BK_TO_CSTMR_STMT      XMLR4       VARCHAR       35       1
MSG_CRE_DT_TM                   BK_TO_CSTMR_STMT      XMLR4       TIMESTMP      12       2
BK_TO_CSTMR_STMT                BK_TO_CSTMR_STMT      XMLR4       XML           14       3
DB2_GENERATED_DOCID_FOR_XML     BK_TO_CSTMR_STMT      XMLR4       BIGINT         8       4
DOCID                           XBK_TO_CSTMR_STMT     XMLR4       BIGINT         8       1
MIN_NODEID                      XBK_TO_CSTMR_STMT     XMLR4       VARBIN       128       2
XMLDATA                         XBK_TO_CSTMR_STMT     XMLR4       VARBIN     15850       3
START_TS                        XBK_TO_CSTMR_STMT     XMLR4       BINARY         8       4
END_TS                          XBK_TO_CSTMR_STMT     XMLR4       BINARY         8       5
DSNE610I NUMBER OF ROWS DISPLAYED IS 4
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+--
```

Example 4-5 describes the following tasks:

- The relationship between the base table and XML table is described in the catalog table SYSIBM.SYSXMLRELS and this information can be retrieved as shown in **Query 6**. The XMLRELOBID column shows internal identifier of the relationship between the base table and the XML table.
- Information about the DOCID and NODEID indexes can be retrieved from the catalog table SYSIBM.SYSINDEXES as shown in **Query 7**.

  Column IX_EXTENSION_TYPE shows N in the first row, indicating that this row is for the NODEID index, which is an extended index. Column IX_EXTENSION_TYPE has blanks in the second row, indicating that this row is for the DOCID index, which is a simple index.

*Example 4-5 Catalog Queries (3 of 3)*

```
-- Query 6

SELECT SUBSTR(TBOWNER,1,10) AS TBOWNER,
       SUBSTR(TBNAME,1,20) AS TBNAME,
       SUBSTR(COLNAME,1,20) AS COLNAME,
       SUBSTR(XMLTBOWNER,1,10) AS XMLTBOWNER,
       SUBSTR(XMLTBNAME,1,20) AS XMLTBNAME,
       XMLRELOBID
FROM SYSIBM.SYSXMLRELS
WHERE TBOWNER = USER
  AND TBNAME = 'BK_TO_CSTMR_STMT';
---------+---------+---------+---------+---------+---------+---------+--------+--------+
TBOWNER     TBNAME            COLNAME            XMLTBOWNER  XMLTBNAME          XMLRELOBID
---------+---------+---------+---------+---------+---------+---------+--------+--------+
XMLR4       BK_TO_CSTMR_STMT BK_TO_CSTMR_STMT   XMLR4       XBK_TO_CSTMR_STMT           7
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
--Query 7

SELECT SUBSTR(NAME,1,30) AS NAME,
       SUBSTR(CREATOR,1,10) AS CREATOR,
       SUBSTR(TBNAME,1,20) AS TBNAME,
       SUBSTR(TBCREATOR,1,10) AS TBCREATOR,
       SUBSTR(DBNAME,1,10) AS DBNAME,
       SUBSTR(INDEXSPACE,1,20) AS INDEXSPACE,
       IX_EXTENSION_TYPE AS IXET
FROM SYSIBM.SYSINDEXES
WHERE DBNAME = 'DSN00242'
  AND CREATOR = USER ;
---------+---------+---------+---------+---------+---------+---------+---------+---------+--
NAME                           CREATOR   TBNAME              TBCREATOR  DBNAME     INDEXSPACE  IXET
---------+---------+---------+---------+---------+---------+---------+---------+---------+--
I_NODEIDXBK_TO_CSTMR_STMT      XMLR4     XBK_TO_CSTMR_STMT   XMLR4      DSN00242   IRNODEID    N
I_DOCIDBK_TO_CSTMR_STMT        XMLR4     BK_TO_CSTMR_STMT    XMLR4      DSN00242   IRDOCIDB
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
```

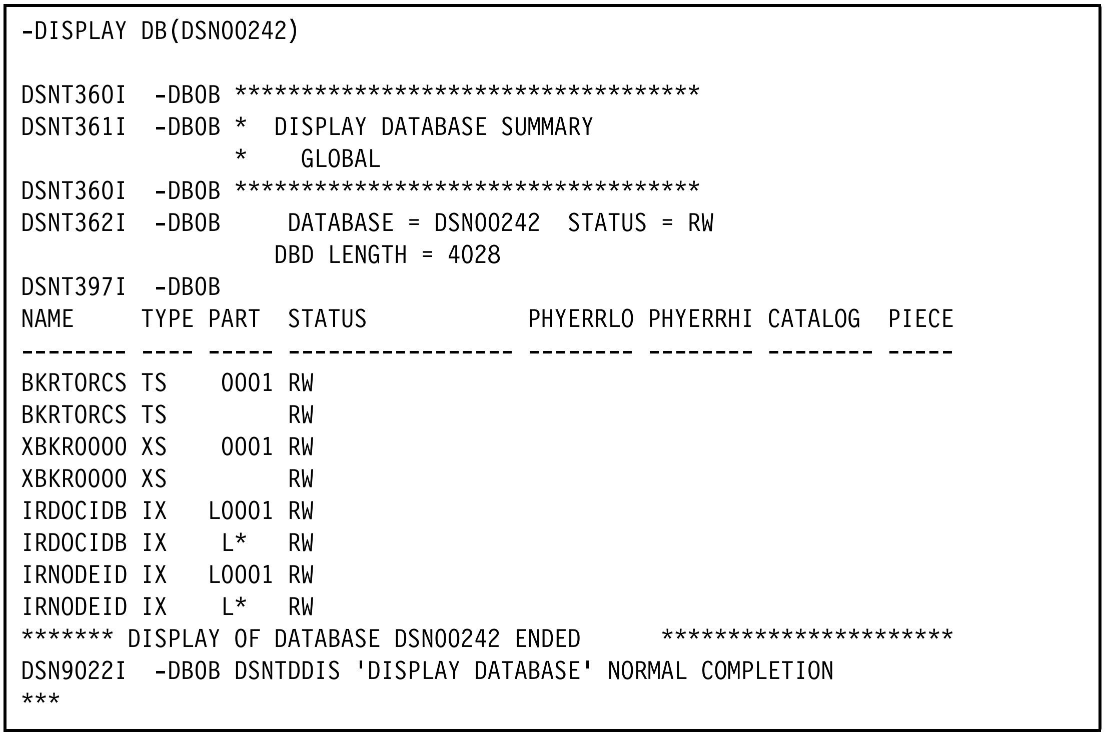
## 4.5 Display database command

If you issue the DB2 -DISPLAY DATABASE command, you can see all the base objects and the XML objects that are associated with the database as shown in Figure 4-9.

TYPE is TS for a table space, IX for an index space, and XS for an XML table space.

PART is the partition number. Because we have one partition for the partition-by-growth table space for both the base table and the XML table, PART shows 00001. Over time, the table space can grow to more partitions and that is why there are two lines for each of the base and XML table spaces, at present with no value.

For non-partitioned indexes, it is the logical partition number preceded by the character L (for example, L0001). This is the case for the DOCID and NODEID indexes.



```
-DISPLAY DB(DSN00242)

DSNT360I  -DB0B ***********************************
DSNT361I  -DB0B *  DISPLAY DATABASE SUMMARY
                *    GLOBAL
DSNT360I  -DB0B ***********************************
DSNT362I  -DB0B     DATABASE = DSN00242  STATUS = RW
                   DBD LENGTH = 4028
DSNT397I  -DB0B
NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
BKRTORCS TS    0001 RW
BKRTORCS TS         RW
XBKR0000 XS    0001 RW
XBKR0000 XS         RW
IRDOCIDB IX   L0001 RW
IRDOCIDB IX    L*   RW
IRNODEID IX   L0001 RW
IRNODEID IX    L*   RW
******* DISPLAY OF DATABASE DSN00242 ENDED      *********************
DSN9022I  -DB0B DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
***
```

*Figure 4-9 -DISPLAY DATABASE command output*

## 4.6 Ingesting XML data

You can use various techniques to ingest XML data. Various forms of the INSERT statement and the LOAD utility exist.

In this section, we show an example of the SQL INSERT statement that uses a string literal to insert rows into a table that contains XML columns. This form of INSERT is suitable for small documents. The host variable or file reference versions of INSERT are applicable for any length.

Example 4-6 on page 70 shows the successful insertion of the shorter version of the Message Received XML document for our application scenario shown in Example A-2 on page 274.

*Example 4-6 Using the SQL INSERT statement to insert XML document to an XML column*

```
INSERT INTO XMLR4.BK_TO_CSTMR_STMT(BK_TO_CSTMR_STMT)
values('<?xml version="1.0" encoding="UTF-8" ?>
  <Document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02">
    <BkToCstmrStmt>
      <GrpHdr>
         <MsgId>AAAASESS-FP-STAT001</MsgId>
         <CreDtTm>2010-10-18T17:00:00+01:00</CreDtTm>
      </GrpHdr>
      <Stmt>
 <Id>AAAASESS-FP-STAT001</Id>
          <CreDtTm>2010-10-18T17:00:00+01:00</CreDtTm>
            <FrToDt>
              <FrDtTm>2010-10-18T08:00:00+01:00</FrDtTm>
              <ToDtTm>2010-10-18T17:00:00+01:00</ToDtTm>
            </FrToDt>
          <Acct>
              <Id>
                <Othr>
                 <Id>50000000054910000003</Id>
                </Othr>
             </Id>
             <Ownr>
               <Nm>FINPETROL</Nm>
             </Ownr>
           <Svcr>
               <FinInstnId>
                  <Nm>AAAA BANKEN</Nm>
               </FinInstnId>
             </Svcr>
            </Acct>
            <Bal>
              <Tp>
          <CdOrPrtry>
                  <Cd>OPBD</Cd>
                </CdOrPrtry>
              </Tp>
              <Amt Ccy="SEK">500000</Amt>
              <CdtDbtInd>CRDT</CdtDbtInd>
            </Bal>
            <Bal>
             <Tp>
               <CdOrPrtry>
                 <Cd>CLBD</Cd>
               </CdOrPrtry>
             </Tp>
             <Amt Ccy="SEK">435678.50</Amt>
             <CdtDbtInd>CRDT</CdtDbtInd>
           </Bal>
           <Ntry>
         <Amt Ccy="SEK">105678.50</Amt>
             <BookgDt>
              <DtTm>2010-10-18T13:15:00+01:00</DtTm>
             </BookgDt>
             <AcctSvcrRef>AAAASESS-FP-CN_98765/01</AcctSvcrRef>
           </Ntry>
          <Ntry>
              <Amt Ccy="SEK">200000</Amt>
              <BookgDt>
               <DtTm>2010-10-18T10:15:00+01:00</DtTm>
              </BookgDt>
              <AcctSvcrRef>AAAASESS-FP-ACCR-01</AcctSvcrRef>
            </Ntry>
            <Ntry>
              <Amt Ccy="SEK">30000</Amt>
          <BookgDt>
              <DtTm>2010-10-18T15:15:00+01:00</DtTm>
        </BookgDt>
           <AcctSvcrRef>AAAASESS-FP-CONF-FX</AcctSvcrRef>
         </Ntry>
    </Stmt>
    <Stmt>
        <Id>AAAASESS-FP-STAT002</Id>
        <CreDtTm>2010-10-17T17:00:00+01:00</CreDtTm>
         <FrToDt>
           <FrDtTm>2010-10-17T08:00:00+01:00</FrDtTm>
           <ToDtTm>2010-10-17T17:00:00+01:00</ToDtTm>
         </FrToDt>
        <Acct>
          <Id>
            <Othr>
              <Id>50000000054910000004</Id>
      </Othr>
           </Id>
         <Ownr>
              <Nm>FINPETROL</Nm>
            </Ownr>
            <Svcr>
              <FinInstnId>
                <Nm>AAAB BANKEN</Nm>
              </FinInstnId>
            </Svcr>
          </Acct>
          <Bal>
            <Tp>
              <CdOrPrtry>
                <Cd>OPAV</Cd>
              </CdOrPrtry>
            </Tp>
         <Amt Ccy="SEK">500300</Amt>
             <CdtDbtInd>CRDT</CdtDbtInd>
        </Bal>
           <Bal>
             <Tp>
               <CdOrPrtry>
                 <Cd>FWAV</Cd>
               </CdOrPrtry>
             </Tp>
             <Amt Ccy="SEK">435478.50</Amt>
```

```
              <CdtDbtInd>CRDT</CdtDbtInd>
           </Bal>
          <Ntry>
              <Amt Ccy="SEK">105378.50</Amt>
              <CdtDbtInd>CRDT</CdtDbtInd>
              <ValDt>
               <Dt>2010-10-17</Dt>
              </ValDt>
              <AcctSvcrRef>AAAASESS-FP-CN_98764/01</AcctSvc
          </Ntry>
          <Ntry>
         <Amt Ccy="SEK">200100</Amt>
             <ValDt>
              <Dt>2010-10-17</Dt>
             </ValDt>
             <AcctSvcrRef>AAAASESS-FP-ACCR-02</AcctSvcrRef>
          </Ntry>
          <Ntry>
             <Amt Ccy="SEK">30020</Amt>
             <BookgDt>
              <DtTm>2010-10-17T15:15:00+01:00</DtTm>
             </BookgDt>
             <AcctSvcrRef>AAAASESS-FP-CONF-FY</AcctSvcrRef>
          </Ntry>
      </Stmt>
    </BkToCstmrStmt>
   </Document>');
---------------------------------------------------------------------------------
DSNE615I NUMBER OF ROWS AFFECTED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
```

This technique is cumbersome for large XML documents. We show how to use the LOAD utility in such cases. LOAD utility is discussed in Chapter 9, “Utilities with XML” on page 183.

## 4.7 XML indexes

An XML index can be used to improve the efficiency of queries on XML documents that are stored in an XML column.

The same way that you define relational indexes on selected columns of a relational table, you define XML indexes on selected elements and attributes within a single XML column of a table. In particular, XML indexes in DB2 do not automatically index all the values in an XML column, but only those that you choose. Although you may choose to index all elements and attributes, you should typically index only those elements and attributes that are frequently used in predicates and join conditions.

Rather than providing access to the beginning of a document, index entries in an XML index provide access to nodes within the document by creating index keys based on XML pattern expressions. Because multiple parts of a XML document can satisfy an XML pattern, DB2 might generate multiple index keys when it inserts values for a single document into the index.

You create an XML index by using the CREATE INDEX statement, and drop an XML index using the DROP INDEX statement. The GENERATE KEY USING XMLPATTERN clause you include with the CREATE INDEX statement specifies what you want to index.

Several keywords that are used with the CREATE INDEX statement for indexes on non-XML columns do not apply to indexes over XML data.

Example 4-7 shows an the creation of an index on the sample XML document for the application scenario.

*Example 4-7 XML index on DtTm elements*

```
CREATE INDEX IXMLNTRY
ON BK_TO_CSTMR_STMT(BK_TO_CSTMR_STMT) 1
GENERATE KEY USING XMLPATTERN 2
'declare default element namespace
  "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
/Document/BkToCstmrStmt/Stmt/Ntry/BookgDt/DtTm'
AS SQL TIMESTAMP 3
```

Note the following information in Example 4-7:

1. The XML index is defined on the BK_TO_CSTMR_STMT column of the BK_TO_CSTMR_STMT table. BK_TO_CSTMR_STMT must be of the XML data type.
2. The GENERATE KEY USING XMLPATTERN clause provides information about what you want to index. This clause is called an *XML index specification*. The XML index specification contains an XML pattern clause. The XML pattern clause in this example indicates that you want to index the values of the DtTm element. The namespace declaration is necessary because the XML documents have a namespace declaration.
3. The AS SQL TIMESTAMP indicates that indexed values are stored as TIMESTAMP values.

Only one index specification clause is allowed in a CREATE INDEX statement. However, you may create multiple XML indexes on an XML column.

Every XML pattern expression that you specify in a CREATE INDEX statement must be associated with a data type. The data type must be VARCHAR, DECFLOAT, DATE, or TIMESTAMP.

You can interpret the result of pattern expression as multiple data types. For example, the value 123 has a character representation, but it can also be interpreted as the number 123. You may create separate indexes on the same pattern expression with separate data types, so that the data can be indexed, regardless of its data type.

If you validate your XML documents against an XML schema, ensure that the data type specifications in the XML schema match the data types that you use for your indexes.

The UNIQUE keyword in XML index definitions has a similar but slightly different meaning than it does for relational index definitions:

- For relational indexes, the UNIQUE keyword in the CREATE INDEX statement enforces uniqueness across all rows in a table.
- For indexes over XML data, the UNIQUE keyword enforces uniqueness across all documents in an XML column.

For an XML index, DB2 enforces uniqueness for the following items:

- The data type of the index
- The XML path to a node
- The value of the node after the XML value has been cast to the SQL data type that is specified for the index

Because rounding can occur during conversion of an index key value to the specified data type for the index, multiple values that appear to be unique in the XML document might rarely result in duplicate key errors.

# 5

# Validating XML data

In this chapter, we provide details about user-controlled and automatic validation of XML documents.

This chapter contains the following topics:

- XML schema validation
- XML type modifier
- Automatic validation
- User-controlled validation
- Determining whether an XML document has been validated

© Copyright IBM Corp. 2011. All rights reserved.

## 5.1 XML schema validation

XML schema validation is the process of determining whether the structure, content, and data types of an XML document are valid according to an XML schema.

In addition, XML schema validation strips the ignorable white space from the input document.

The two ways that you can validate an XML document in DB2 10 are as follows:

- Automatically, by including an XML type modifier in the XML column definition in a CREATE TABLE or ALTER TABLE statement

  When a column has an XML type modifier, DB2 implicitly validates documents that are inserted into the column or when documents in the column are updated.

- User-controlled, by executing the DSN_XMLVALIDATE built-in function when you insert a document into an XML column or update a document in an XML column or before selecting back (not necessarily into a table)

Validation is optional when you insert data into an XML column with no XML type modifier. Validation is mandatory when you insert data into an XML column with an XML type modifier.

This chapter describes both methods (automatic and user-controlled). First, we describe the XML type modifier.

## 5.2 XML type modifier

Automatic validation requires XML type modifier.

The XML data type can accept any well-formed XML documents. However, in many cases, users want to store, in one XML column, the documents that have similar structures or conform to the same XML schema. DB2 10 introduces the *XML type modifier*, which qualifies the XML data type with a set of one or more XML schemas. The value of an XML column with an XML type modifier must conform to at least one XML schema that is specified in the type modifier.

When you define an XML column, you may add an XML type modifier. An XML type modifier associates a set of one or more XML schemas with the XML data type. You may use an XML type modifier to cause all XML documents that are stored in an XML column to be validated according to one of the XML schemas that is specified in the type modifier.

The XML type modifier can identify more than one XML schema. You might want to associate more than one XML schema with an XML type modifier for the following reasons:

- The requirements for an XML schema evolve over time.

  An XML column might contain documents that describe only one type of information, but certain fields in newer documents might need to be different from fields in the older documents. As new document versions are required, you can add new XML schemas to the XML type modifier.

- A single XML column contains XML documents of various kinds.

  An XML column might contain documents that have several formats. In this case, each type of document needs its own XML schema.

Alternatively, you might want to associate a single XML schema with multiple type modifiers. An XML schema can define many various documents. You might need to separate the XML documents into different columns, but specify the same XML schema in a type modifier for each column.

For example, a sales department might have one XML schema that defines purchase orders and billing statements. You might store purchase orders in one XML column, and billing statements in another XML column. Both XML columns have an XML type modifier that points to the same XML schema, but each column restricts documents with separate root elements in the XML schema.

You define an XML type modifier in a CREATE TABLE or ALTER TABLE statement as part of an XML column definition.

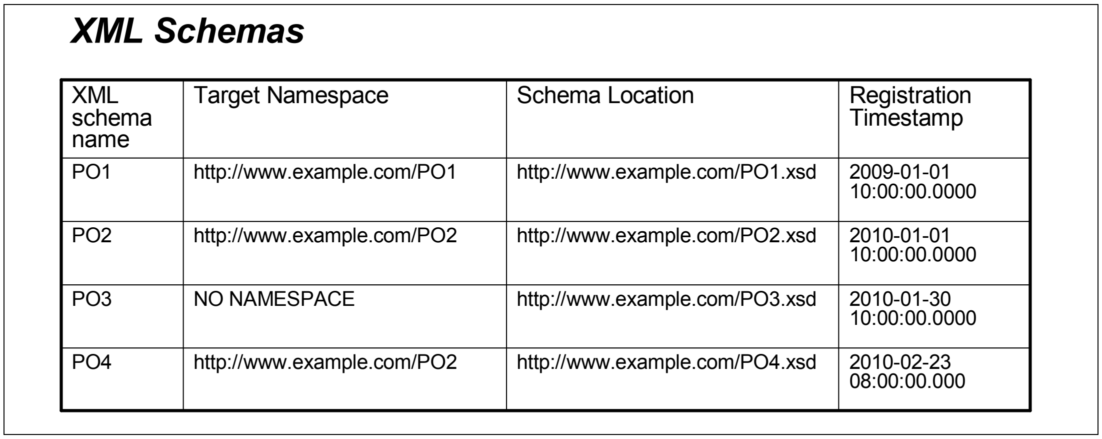
Not all XML schemas that the XML type modifier identifies must be registered before you execute the CREATE or ALTER statement. If the XML type modifier specifies a target namespace, only the XML schemas in that target namespace that exist when the CREATE or ALTER statement is executed are associated with the XML type modifier.

If altered data type is XML, the old data type of the altered column must also be XML:

- If the old data type has no XML type modifier and the new data type does, ensure that all values in the XML column are valid according to the XML schema that is specified in the type modifier. The XML table space for the column that is being changed is left in CHECK-pending status.
- If the old data type has the XML type modifier but the new data type has no type modifier, the existing values do not need to be revalidated. The state of the table space is not changed.
- If the XML schemas that are specified in the old XML type modifier are a subset of the XML schemas that are specified in the new XML type modifier, the existing values do not need to be revalidated. The state of the XML table space is not changed.
- If the XML schemas that are specified in the old XML type modifier are *not* a subset of the XML schemas that are specified in the new XML type modifier, the XML table space for the column that is being changed is left in the CHECK-pending status.

Changing an XML column to use a different type modifier does not result in the invalidation of dependent plans, packages, or statements in the dynamic statement cache. Also, changing an XML column to use a different type modifier does not generate a new version of the table.

Figure 5-1 shows the XML schemas that the examples in this section refer to for defining an XML type modifier.



**XML Schemas**

| XML schema name | Target Namespace | Schema Location | Registration Timestamp |
|---|---|---|---|
| PO1 | http://www.example.com/PO1 | http://www.example.com/PO1.xsd | 2009-01-01 10:00:00.0000 |
| PO2 | http://www.example.com/PO2 | http://www.example.com/PO2.xsd | 2010-01-01 10:00:00.0000 |
| PO3 | NO NAMESPACE | http://www.example.com/PO3.xsd | 2010-01-30 10:00:00.0000 |
| PO4 | http://www.example.com/PO2 | http://www.example.com/PO4.xsd | 2010-02-23 08:00:00.000 |

*Figure 5-1 XML schemas*

Example 5-1 shows how to specify an XML type modifier for an XML column at creation time.

*Example 5-1 Specify an XML type modifier for an XML column at creation time*

```
CREATE TABLE PURCHASEORDERS(
  ID INT NOT NULL,
  CONTENT XML(XMLSCHEMA ID SYSXSR.PO1))
```

A table for purchase orders contains an XML column named CONTENT. The documents in the XML column must be validated according to XML schema SYSXSR.PO1, which has already been registered.

To alter an existing XML column to include an XML type modifier or remove an XML type modifier, use ALTER TABLE statement.

Example 5-2 shows the table definition without XML type modifier specified.

*Example 5-2 Table definition without XML type modifier*

```
CREATE TABLE PURCHASEORDERS(
  ID INT NOT NULL,
  CONTENT XML)
```

The table contains several XML documents. The documents in the XML column must be validated according to the XML schema SYSXSR.PO1, which has already been registered. Alter the XML column to add an XML type modifier that specifies SYSXSR.PO1, as shown in Example 5-3.

*Example 5-3 Specify XML type modifier for XML column at alteration time*

```
ALTER TABLE PURCHASEORDERS
  ALTER CONTENT
  SET DATA TYPE XML(XMLSCHEMA ID SYSXSR.PO1)
```

**Note:** The table space that contains the XML documents for the CONTENT column is put in CHECK-pending status.

You may add an XML schema to the XML type modifier. See Example 5-4. Suppose PO2 is a new version of the purchase order schema. You may use the ALTER TABLE statement to reset the XML type modifier of the CONTENT column to include both PO1 and PO2.

*Example 5-4 Add an XML schema to the XML type modifier*

```
ALTER TABLE PURCHASEORDERS
  ALTER CONTENT
  SET DATA TYPE XML(XMLSCHEMA ID SYSXSR.PO1, ID SYSXSR.PO2)
```

**XML schema:** Because the XML schema specified in the old type modifier is a subset of the new type modifier, the existing values of the CONTENT column do not need to be validated again. Thus, the state of the XML table space for the CONTENT column stays unchanged. If the XML schema that is specified in the old XML type modifier is *not* a subset of the XML schema that is specified in the new XML type modifier, the XML table space for the column that is being changed remains in the CHECK-pending status.

You may also reset the data type of CONTENT to XML without type modifier.

Example 5-5 shows how to reset XML type modifier for an XML column at alter time.

*Example 5-5 Reset XML type modifier for XML column at alter time*

```
ALTER TABLE PURCHASEORDERS
  ALTER CONTENT
  SET DATA TYPE XML
```

**CONTENT column in this example:** The existing values of the CONTENT column do not need to be validated again.

Validation is automatically performed for INSERT and UPDATE SQL statements and the LOAD utility if the XML column is defined with the XML type modifier.

Catalog table support for XML type modifiers are as follows:

- SYSIBM.SYSXMLTYPMOD contains rows for the XML type modifiers.
- SYSIBM.SYSXMLTYPMSCHEMA contains a row for each XML schema specification for one XML type modifier.

Instead of specifying the schema name directly as shown in all the examples, you may also specify the URI and LOCATION keywords, so the schema name can be derived.

Example 5-6 shows how to specify the schema location hint. Both PO2 and PO4 have the same target namespace `http://www.example.com/PO2`. If you want to use PO2, you may add `LOCATION ‘http://www.example.com/PO2.xsd’` after the `URI ‘http://www.example.com/PO2’` clause.

*Example 5-6 Identify an XML schema by target namespace and schema location*

```
CREATE TABLE PURCHASEORDERS(
  ID INT NOT NULL,
  CONTENT XML(XMLSCHEMA URI ‘http://www.example.com/PO2’
                      LOCATION ‘http://www.example.com/PO2.xsd’ ))
```

Example 5-7 shows that XML type modifier uses only the URI keyword to identify the XML schema.

*Example 5-7 Identify an XML schema by target namespace*

```
CREATE TABLE PURCHASEORDERS(
  ID INT NOT NULL,
  CONTENT XML(XMLSCHEMA URI ‘http://www.example.com/PO2’))
```

**Using the URI keyword:** If you execute the CREATE TABLE statement before PO4 is registered, only PO2 is added to the type modifier in SYSIBM.SYSXMLTYPMSCHEMA. When PO4 is registered later, the XML type modifier for the CONTENT column remains unchanged. If you execute the CREATE TABLE statement after PO4 is registered, an SQL error occurs because the XML type modifier uses the URI keyword to identify two XML schemas PO2 and PO4. The URI keyword must identify only one XML schema.

If an XML schema does not contain the targetNamespace attribute in its schema document, it can be referenced in the XML type modifier by NO NAMESPACE.

In Example 5-8, DB2 chooses PO3 as the XML type modifier.

*Example 5-8 No namespace*

```
CREATE TABLE PURCHASEORDERS(
  ID INT NOT NULL,
  CONTENT XML(XMLSCHEMA NO NAMESPACE
                      LOCATION ‘http://www.example.com/PO3.xsd’ ))
```

If an XML schema has more than one global element declaration and you want to validate the XML value against one of them, you can specify the ELEMENT clause. Assume that the purchase order schema has declared two global elements: purchaseOrder and comment. Therefore, a document whose root element is either purchaseOrder or comment could be valid according to PO1 and can be stored in the PURCHASEORDERS table. However, this approach might not be desirable. If you want to store only purchase order documents in the CONTENT column, you can specify the ELEMENT “`purchaseOrder`” in the XML type modifier for CONTENT. Example 5-9 shows how you can do this step.

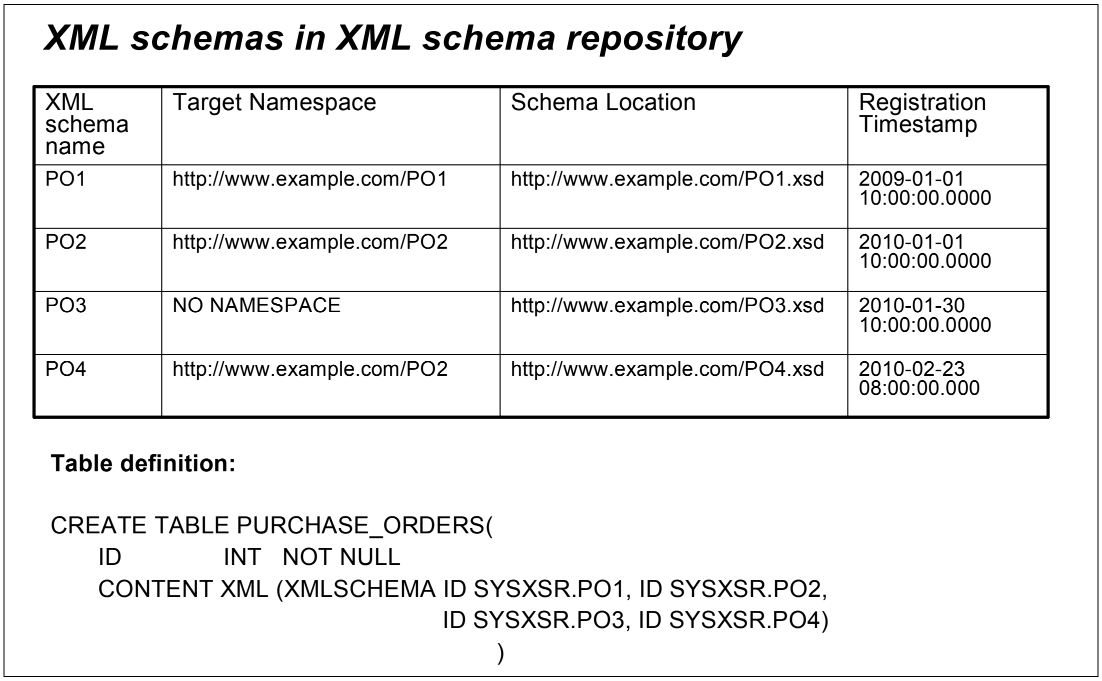
*Example 5-9 Specifying global element name*

```
CREATE TABLE PURCHASEORDERS(
  ID INT NOT NULL,
  CONTENT XML(XMLSCHEMA ID SYSXSR.PO1
                      ELEMENT “purchaseOrder”))
```

## 5.3 Automatic validation

You may automate XML schema validation by adding an XML type modifier to an XML column definition. Before schema validation through an XML type modifier can occur, all schema documents that make up an XML schema must be registered in the built-in XML schema repository (XSR).

Figure 5-2 on page 81 shows the table definition and the entries in the XSR for the schemas used as XML type modifiers in the table definition.



## XML schemas in XML schema repository

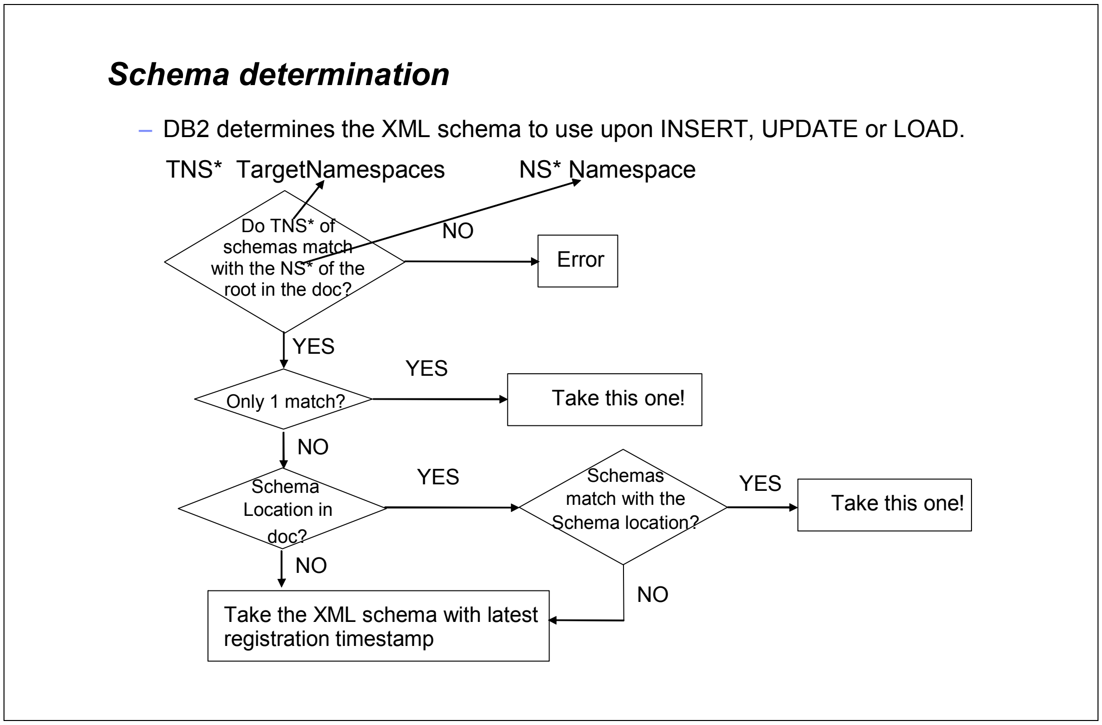
| XML schema name | Target Namespace | Schema Location | Registration Timestamp |
|---|---|---|---|
| PO1 | http://www.example.com/PO1 | http://www.example.com/PO1.xsd | 2009-01-01 10:00:00.0000 |
| PO2 | http://www.example.com/PO2 | http://www.example.com/PO2.xsd | 2010-01-01 10:00:00.0000 |
| PO3 | NO NAMESPACE | http://www.example.com/PO3.xsd | 2010-01-30 10:00:00.0000 |
| PO4 | http://www.example.com/PO2 | http://www.example.com/PO4.xsd | 2010-02-23 08:00:00.000 |

**Table definition:**

```
CREATE TABLE PURCHASE_ORDERS(
    ID          INT   NOT NULL
    CONTENT XML (XMLSCHEMA ID SYSXSR.PO1, ID SYSXSR.PO2,
                           ID SYSXSR.PO3, ID SYSXSR.PO4)
                                 )
```

*Figure 5-2 XML Schemas in XML schema repository*

Figure 5-3 shows how DB2 chooses an XML schema when the XML type modifier specifies multiple schemas.



## Schema determination

- DB2 determines the XML schema to use upon INSERT, UPDATE or LOAD.

TNS* TargetNamespaces
NS* Namespace
Do TNS* of schemas match with the NS* of the root in the doc?
NO
Error
YES
Only 1 match?
YES
Take this one!
NO
Schema Location in doc?
YES
Schemas match with the Schema location?
YES
Take this one!
NO
Take the XML schema with latest registration timestamp
NO

*Figure 5-3 Schema determination*

You may include more than one XML schema in an XML type modifier. When you insert into or update an XML column, DB2 chooses one XML schema to do validation.

DB2 uses the following process to determine which XML schema to use.

- If the operation is an update operation, and an XML schema that is specified by the XML type modifier has already been used to validate the original document, DB2 uses the same XML schema to validate the updated document.
- If there is only one XML schema whose target namespace matches the namespace name of the root element node in the document that is being validated (the XML instance document), DB2 chooses that XML schema to validate the XML document.
- If there is more than one XML schema with a target namespace that matches the namespace name of the root element, DB2 chooses an XML schema by using the schema location hint. The root element node of an XML instance document can contain an `xsi:schemaLocation` attribute. That attribute consists of one or more pairs of URI references, separated by white space. The first member of each pair is a namespace name, and the second member of the pair is a URI that describes where to find an appropriate schema document for that namespace. The second member of each pair is the schema location hint for the namespace name that is specified in the first member.

For example, a schema location attribute is as follows:

```
xsi:schemaLocation="http://www.example.com/PO2 http://www.example.com/PO4.xsd"
```

The first member of the pair, `http://www.example.com/PO2`, is the namespace name. The second member of the pair, `http://www.example.com/PO4.xsd`, is the URI that provides the schema location hint.

DB2 uses the schema location hint to choose an XML schema in the following way:

- If the root element node contains an `xsi:schemaLocation` attribute, DB2 searches the attribute value for a schema location hint with a corresponding namespace name that matches the namespace name in the root element node.
- If DB2 finds a schema location hint, DB2 uses the hint to identify an XML schema whose schema location URI is identical to the schema location hint. DB2 validates the input document against that schema.
- If the root element does not contain an `xsi:schemaLocation` attribute, or the `xsi:schemaLocation` attribute does not contain a schema location hint with a corresponding namespace name that matches the namespace name in the root element node, DB2 uses the XML schema with the same target namespace and the latest registration time stamp.

Examples of how DB2 determines the schema to be used for validation from an XML type modifier are listed next.

In Example 5-10, DB2 chooses XML schema PO1.

*Example 5-10 Schema selection for validation from an XML type modifier (1 of 4)*

```
INSERT INTO PURCHASE_ORDERS VALUES(1,
‘<po:purchaseOrder xmlns:po="http://www.example.com/PO1">
...
</po:purchaseOrder>‘);
```

The namespace name in the root element of the instance document is as follows:

```
http://www.example.com/PO1
```

This name matches only the target namespace for XML schema PO1.

In Example 5-11, DB2 chooses XML schemas PO2 and PO4 in this order.

*Example 5-11 Schema selection for validation from an XML type modifier (2 of 4)*

```
INSERT INTO PURCHASE_ORDERS VALUES(2,
‘<po:purchaseOrder xmlns:po="http://www.example.com/PO2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com/PO2
http://www.example.com/PO2.xsd">
...
</po:purchaseOrder>‘);
```

The namespace name in the root element in the instance document is as follows:

```
http:// www.example.com/PO2
```

It matches the target namespace of XML schemas PO2 and PO4. The root element of the instance document also contains an `xsi:schemaLocation` attribute, which has a value that provides the schema location hint:

```
http:// www.example.com/PO2.xsd
```

The schema location hint matches the schema location for XML schema PO2. Therefore DB2 chooses PO2 to validate the instance document. If validation with PO2 fails, DB2 uses PO4.

In Example 5-12, DB2 chooses XML schemas PO4 and PO2 in this order.

*Example 5-12 Schema selection for validation from an XML type modifier (3 of 4)*

```
INSERT INTO PURCHASE_ORDERS VALUES(3,
‘<po:purchaseOrder xmlns:po="http://www.example.com/PO2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com/PO2
http://www.example.com/PO4.xsd">
...
</po:purchaseOrder>‘);
```

The namespace name in the root element in the instance document is as follows:

```
http:// www.example.com/PO2
```

It matches the target namespace of XML schemas PO2 and PO4. The root element of the instance document also contains an `xsi:schemaLocation` attribute whose value provides the schema location hint:

```
http:// www.example.com/PO4.xsd
```

The schema location hint matches the schema location for XML schema PO4. Therefore DB2 chooses PO4 to validate the instance document. If validation with PO4 fails, DB2 uses PO2.

In Example 5-13, DB2 chooses XML schema PO3.

*Example 5-13 Schema selection for validation from an XML type modifier (4 of 4)*

```
INSERT INTO PURCHASE_ORDERS VALUES(4,
‘<purchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.example.com/PO3.xsd">
...
</purchaseOrder>‘);
```

The root element of the instance document has no namespace name. XML schema PO3 has no target namespace. Therefore, DB2 uses PO3 for validation.

Note that, after you update an XML document in a column that has an XML type modifier, DB2 validates again, all or part of the document:

- If the XML type modifier includes several XML schemas, DB2 uses the same XML schema for validation that is used for the original validation.
- If you update an entire document, DB2 validates the entire document. However, if you use the XMLMODIFY[1] function to update only a portion of the document, DB2 might need to validate only the updated portion.

## 5.4 User-controlled validation

One other way to do XML schema validation is by executing the following built-in function:

```
SYSIBM.DSN_XMLVALIDATE
```

Before you can invoke DSN_XMLVALIDATE, all schema documents that make up an XML schema must be registered in the XML schema repository and successfully compiled.

You may use DSN_XMLVALIDATE with a type modifier, which can be used to override a schema selected by DB2. DB2 checks whether the schema used in DSN_XMLVALIDATE is one of the schemas in the type modifier, and skips the double validation.

Several forms of DSN_XMLVALIDATE are available:

```
DSN_XMLVALIDATE(string-expression)
DSN_XMLVALIDATE(xml-expression)
DSN_XMLVALIDATE(string-expression, varchar-expression)
DSN_XMLVALIDATE(xml-expression, varchar-expression)
DSN_XMLVALIDATE(string-expression1, string-expression2, string-expression3)
DSN_XMLVALIDATE(xml-expression1, string-expression2, string-expression3)
```

For all forms, the first parameter contains the document that you want to validate. Note the following information:

- For forms with one parameter, the target namespace and optional schema location of the XML schema must be in the root element of the instance document that you want to validate.
- For forms with two parameters, the second parameter is the name of the schema object to use for validation of the document. That object must be registered in the XML schema repository.
- For forms with three parameters, the second and third parameter contain the names of a namespace URI and a schema location hint that identify the XML schema object to use for validation of the document. That object must be registered in the XML schema repository.

XML schemas that are used for validation are the same as shown in Figure 5-2 on page 81. However, the following CREATE TABLE statement is used to create the table:

```
CREATE TABLE PURCHASE_ORDERS(
  ID      INT   NOT NULL
  CONTENT XML )
```

[1] The XMLMODIFY function returns an XML value that might have been modified by the evaluation of an XPath updating-expression and XPath variables that are specified as input arguments.

Examples of the criteria for how DB2 chooses XML schema for DSN_XMLVALIDATE are listed next.

In Example 5-14, DB2 chooses XML schema PO1.

*Example 5-14 Schema selection for validation for DSN_XMLVALIDATE (1 of 4)*

```
INSERT INTO PURCHASE_ORDERS VALUES(1,
DSN_XMLVALIDATE(‘<po:purchaseOrder xmlns:po="http://www.example.com/PO1">
...
</po:purchaseOrder>‘));
```

The DSN_XMLVALIDATE invocation does not specify an XML schema or target namespace and schema location hint, therefore DB2 uses the information in the instance document.

The namespace name in the root element of the instance document is as follows:

```
http://www.example.com/PO1
```

This name matches only the target namespace for XML schema PO1.

In Example 5-15, DB2 chooses XML schema PO2.

*Example 5-15 Schema selection for validation for DSN_XMLVALIDATE (2 of 4)*

```
INSERT INTO PURCHASE_ORDERS VALUES(2,
DSN_XMLVALIDATE(‘<po:purchaseOrder xmlns:po="http://www.example.com/PO2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com/PO2
http://www.example.com/PO2.xsd">
...
</po:purchaseOrder>‘,’SYSXSR.PO2’));
```

The DSN_XMLVALIDATE invocation specifies XML schema SYSXSR.PO2.

In Example 5-16, DB2 chooses XML schema PO4.

*Example 5-16 Schema selection for validation for DSN_XMLVALIDATE (3 of 4)*

```
INSERT INTO PURCHASE_ORDERS VALUES(3,
DSN_XMLVALIDATE(‘<po:purchaseOrder xmlns:po="http://www.example.com/PO2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com/PO2
http://www.example.com/PO4.xsd">
...
</po:purchaseOrder>‘,’http://www.example.com/PO2’));
```

The DSN_XMLVALIDATE invocation specifies the following namespace:

```
http://www.example.com/PO2
```

Two XML schemas, PO2 and PO4, have that target namespace. DB2 uses PO4, because it has the later time stamp.

In Example 5-17, DB2 chooses PO3.

*Example 5-17 Schema selection for validation for DSN_XMLVALIDATE (4 of 4)*

```
INSERT INTO PURCHASE_ORDERS VALUES(4,
DSN_XMLVALIDATE(‘<purchaseOrder
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.example.com/PO3.xsd">
...
</purchaseOrder>‘));
```

The DSN_XMLVALIDATE invocation does not specify an XML schema or target namespace and schema location hint, therefore DB2 uses the information in the instance document. The root element node in the instance document contains an `xsi:noNamespaceSchemaLocation` attribute with the following value:

```
http://www.example.com/PO3.xsd
```

Therefore, DB2 uses XML schema PO3, which has no target namespace, and the following schema location:

```
http://www.example.com/PO3.xsd
```

The two versions of DSN_XMLVALIDATE are as follows:

- A user-defined function
- A built-in function

The user-defined function is deprecated. Now, DB2 uses the built-in function instead, even in DB2 9.

To move from the DSN_XMLVALIDATE user-defined function to the DSN_XMLVALIDATE built-in function, use any of the following steps:

- For applications that invoke DSN_XMLVALIDATE by using SYSFUN.DSN_XMLVALIDATE qualified name, the steps are as follows:

  a. Change the name to SYSIBM.DSN_XMLVALIDATE.

     This change is optional. DB2 drops SYSFUN.DSN_XMLVALIDATE and invalidates packages during migration.

     Automatic rebind operations pick up SYSIBM.DSN_XMLVALIDATE.

  b. Prepare the applications again.

- For applications that invoke DSN_XMLVALIDATE without using the qualified name, you do not need to modify the applications. DB2 uses the SYSIBM.DSN_XMLVALIDATE built-in function automatically.

- Optional: Remove the XMLPARSE function that surrounds DSN_XMLVALIDATE.

  The SYSFUN.DSN_XMLVALIDATE user-defined function must be invoked from within the XMLPARSE function. The SYSIBM.DSN_XMLVALIDATE built-in function does not need to be invoked from within the XMLPARSE function.

**Deprecated function:** The SYSFUN.DSN_XMLVALIDATE user-defined function is deprecated. Use the SYSIBM.DSN_XMLVALIDATE built-in function instead. Even if you explicitly call SYSFUN.DSN_XMLVALIDATE, DB2 runs SYSIBM.DSN_XMLVALIDATE.

## 5.5 Determining whether an XML document has been validated

You may use the SQL XMLXSROBJECTID scalar function to determine whether an XML document that is stored in a table has undergone XML validation, and which XML schema was used to validate that document.

XMLXSROBJECTID returns the XSR object identifier of the XML schema that was used to validate the input XML document. The XSR object identifier corresponds to the XSROBJECTID column in the SYSIBM.XSROBJECTS “catalog” table. After you call XMLXSROBJECTID, you may use the returned value to query SYSIBM.XSROBJECTS for the XML schema information. If the XML document has not been validated, XMLXSROBJECTID returns a value of 0 (zero).

The SQL statement in Example 5-18 calls XMLXSROBJECTID to determine which XML documents in the INFO column of the CUSTOMER table have not been validated. The statement then calls DSN_XMLVALIDATE to validate those documents against XML schema SYSXSR.P01.

*Example 5-18 Search for documents not validated*

```
UPDATE CUSTOMER
SET INFO = DSN_XMLVALIDATE(INFO, 'SYSXSR.P01')
WHERE XMLXSROBJECTID(INFO)=0
```

The SQL statement in Example 5-19 retrieves the XML schema names and target namespaces for the XML schemas that were used to validate XML documents in the INFO column of the CUSTOMER table.

*Example 5-19 Retrieve target namespaces and XML schema names used for validation*

```
SELECT DISTINCT S.XSROBJECTNAME, S.TARGETNAMESPACE
FROM CUSTOMER C, SYSIBM.XSROBJECTS S
WHERE XMLXSROBJECTID(INFO) = S.XSROBJECTID
```

6

# DB2 SQL/XML programming

In this chapter, we provide a wide range of DB2 programming examples for pureXML. Over recent years, the amount of programming that is deployed inside DB2 has been growing with the adoption of facilities such as stored procedures, user defined functions, triggers, and WebSphere MQ integration.

The adoption of pureXML is likely to increase this trend. The ease of handling XML documents in native SQL procedures, compared to external language environments, such as COBOL, means that development productivity can be enhanced by encapsulating XML logic within DB2 procedures and functions, so that external programs have to handle only traditional SQL-based functions.

This chapter contains the following topics:

- Native SQL stored procedures and XML
- Receiving XML messages from MQ
- Audit queries (against logged XML messages)
- SQL/XML query techniques
- User-defined functions with XML
- Triggers with XML
- XML joins
- XML with change data capture tools

© Copyright IBM Corp. 2011. All rights reserved.

## 6.1 Native SQL stored procedures and XML

The book application scenario, XML message logging and auditing, described in Chapter 3, “Application scenario” on page 47, represents a common situation in which XML messages are flowing over a WebSphere MQ enterprise service bus, and you want to capture several messages and store them in DB2 pureXML for auditing purposes.

Native stored procedures are an excellent vehicle for capturing XML messages from messaging infrastructure (such as WebSphere MQ and DataPower®) because they have the following characteristics:

- Productive: With them, you can encapsulate multi-step processes into a single DB2 callable routine, which supports the XML data type explicitly.
- Well suited to handling XML: They can use XML documents as input and output parameters, and manipulate XML without necessarily having to persist it to a DB2 table.
- Efficient: They can parse the incoming XML document once, and re-use it as an XML data type without re-parsing.
- Well connected: They integrate well with MQ.

Native stored procedures become more powerful as a result of their support for XML. Prior to DB2 10, a DB2 stored procedure could only pass individual data elements as parameters. Now, the possibility exists to pass them arrays of data within an XML input parameter, which is a more practical approach to implement large scale processing work within the DB2 subsystem, where it is most efficient.

This section shows examples of XML handling with native SQL stored procedures, and then adds integration with WebSphere MQ.

We use the table definitions shown in Example 6-1 for the initial examples. The sample stored procedures show techniques to validate the data manually, and also automatically, which is why we have two versions of the audit logging table.

*Example 6-1 Tables used for examples*

```
-- create base table for storing ISO20022 XML documents, with validation

CREATE TABLE BK_TO_CSTMR_STMT (
   MSG_ID VARCHAR(35) ,
   MSG_CRE_DT_TM TIMESTAMP,
   BK_TO_CSTMR_STMT XML(XMLSCHEMA ID SYSXSR.SG247915_01) NOT NULL)
IN XMLR3DB.TSAUDIT1 ;

-- create alternate base table for ISO20022 XML documents, without validation

CREATE TABLE BK_TO_CSTMR_STMT_MANUALVALIDATE (
   MSG_ID VARCHAR(35) ,
   MSG_CRE_DT_TM TIMESTAMP,
   BK_TO_CSTMR_STMT XML NOT NULL)
IN XMLR3DB.TSAUDIT2 ;

-- create iso20022 currency lookup table

CREATE TABLE CURRENCY (
   ENTITY VARCHAR(50),
   CURRENCY VARCHAR(50),
   ALPHABETIC_CODE CHAR(3),
   NUMERIC_CODE SMALLINT )
IN XMLR3DB.TSAUDIT3 ;

-- create table for error handling logic to store invalid documents

CREATE TABLE XMLR3.INVALID_DOCS (
   INSERT_TIME TIMESTAMP,
   INSERT_DOC XML,
   ERROR_MESSAGE VARCHAR(1000) )
```

We are also using the ISO 20022 bank-to-customer-statement schema, which we have taken directly from the ISO 20022 website without any alteration. We downloaded the schema to the following location and registered the schema with name SYSXSR.SG247915_01 in the DB2 XSR, by using the commands in Example 6-2:

```
D:\XMLRES\WEEK3\REDXMPL\camt.053.001.02.xsd
```

The commands to register schemas in the XSR are available in two environments:

- z/OS UNIX System Services
- DB2 for Linux, UNIX, and Windows command line processor (CLP)

They are explained in more detail in 2.1.6, “XML schema repository and schema validation” on page 36.

*Example 6-2 Registering the ISO 20022 Bnk_To_Cst_Stmt XML schema*

```
register xmlschema 'redbook_bankstmt.xsd' from
   file://D:\SG247915\camt.053.001.02.xsd as SYSXSR.SG247915_01 ;

   DB20000I  The REGISTER XMLSCHEMA command completed successfully.

complete xmlschema SYSXSR.SG247915_01 ;

   DB20000I  The COMPLETE XMLSCHEMA command completed successfully.
```

Alternatively, in Example 6-2, the **`register xmlschema`** command can have the `complete` clause at the end to complete the registration in the same command.

## 6.1.1 Native SQL stored procedure example

We first show an example of how XML can be used with a native SQL stored procedure. Example 6-3 on page 92 shows a simple stored procedure for registering the ISO 20022 Bnk_To_Cst_Stmt XML schema. It was written to perform the following tasks:

1. Receive an XML document as an input parameter.
2. Validate the incoming XML document against the schema in DB2 XSR.
3. Use various XML functions to extract elements from the incoming XML document.
4. Store the document (and extracted fields) in the DB2 audit logging table.
5. Use XML publishing functions to generate a new XML document.
6. Return the generated XML document as an output parameter.

*Example 6-3 Simple stored procedure for registering the schema*

```
CREATE PROCEDURE STOREXML1 (
   IN V_BANKSTMT XML,
   OUT V_MSG_ID VARCHAR(35),
   OUT V_CREDTTM TIMESTAMP,
   OUT V_MINISTMT XML) 1
      LANGUAGE SQL
      MODIFIES SQL DATA
      DISABLE DEBUG MODE

BEGIN

DECLARE VALIDXML XML ;
DECLARE SQLCODE INTEGER ;

SET VALIDXML = DSN_XMLVALIDATE(V_BANKSTMT, 'SYSXSR.SG247915_01') ; 2

SET V_MSG_ID = (
   xmlcast(xmlquery(
      'declare default element
      namespace"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $d/Document/BkToCstmrStmt/GrpHdr/MsgId'
      passing VALIDXML as "d")
   as varchar(35))); 3

SET V_CREDTTM = (
   xmlcast(xmlquery(
      'declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $d/Document/BkToCstmrStmt/GrpHdr/CreDtTm'
      passing VALIDXML as "d")
   as timestamp)); 4

SET V_MINISTMT = (
   xmlquery('declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $d/Document/BkToCstmrStmt/Stmt'
      passing VALIDXML as "d")) ; 5

INSERT INTO BK_TO_CSTMR_STMT_MANUALVALIDATE (
   MSG_ID , MSG_CRE_DT_TM, BK_TO_CSTMR_STMT)
   values ( V_MSG_ID, V_CREDTTM, VALIDXML ) ; 6

END  !
```

The following notes explain each of the annotated points in the source code for the stored procedure.

1. The parameter list includes one input parameter (data type XML) and three output parameters (data types VARCHAR(35), Timestamp, and XML). The ability to support XML as parameters to a callable routine is new in DB2 10.

2. This statement uses the system-provided DSN_XMLVALIDATE function to validate the input XML document against a schema that is defined in the DB2 XSR, WITHOUT storing the XML document in a DB2 table. (Most use cases show schema validation as part of an SQL insert or update).

3. This statement shows the use of the XMLQUERY function to strip a string data element from the XML document, and then cast it to a relational data type using XMLCAST.

4. This statement shows another example of the XMLQUERY function to strip a different field with data type timestamp from the XML document. (DB2 10 supports both timestamp and timestamp with timezone data types in both relational and XML structures).

5. This statement uses an XMLQUERY function to generate a new XML document from the received XML document, and to include the generated XML document as an output parameter from the stored procedure.

6. This statement stores the received XML document, alongside two stripped data elements in a DB2 table.

One of the strengths of the native SQL stored procedure is the ability to operate on an XML document in memory, without storing it in a DB2 table (as performed in steps 2, 3, 4 and 5). This ability helps the stored procedure to be efficient because it saves making unnecessary write and read operations to a DB2 table.

This stored procedure contains XML data types that make it difficult to test from many of the tools you might be using, because they do not have an easy way to pass XML types to and from the procedure (such as SPUFI, DB2 command line processor and so on). A Java program was used as a test driver to call the stored procedure from a Windows DB2 client: it calls the stored procedure with an XML input parameter, and receives the output parameters, and writes them to an output file. This program is called `Teststorexml1.java`, and is included in the additional materials of this book.

This stored procedure example can be improved in several ways, which we develop in this chapter, as follows:

- No error handling logic is included within the stored procedure.
- The three XMLQUERY operations can be replaced by a single XMLTABLE operation.

## 6.1.2 XML error handling in native SQL procedures

The principles of error handling for XML processing is no different from normal native SQL stored procedures. There are many SQLCODES and SQLSTATES that provide diagnostic information in the event of an error relating to the various XML-related functions and procedures in DB2. These error conditions can be handled in the same way as relational errors. You only need to be aware of new situations where errors might occur, and code for them.

The basic approach to error handling in native stored procedures is as follows:

1. Anticipate which error conditions are likely to be encountered during normal use of the system you are building. For example, if you are receiving XML documents from external sources, you must consider that some might be badly formed or fail XML schema validation.
2. Define error conditions that are likely to occur.
3. Define error-handling routines that take the appropriate action for an anticipated circumstance (and a catch-all routine for any other error). For example, if a received XML document fails validation, you might want to store it as a character large object (CLOB) in a separate table for inspection.

The previous SQL-stored procedure in Example 6-3 on page 92 has been modified to include an XMLTABLE function (to replace the two separate XMLQUERY functions) and to include basic error handling. The modified procedure is listed in Example 6-4 on page 94.

*Example 6-4   Stored procedure with error handling logic*

```
CREATE PROCEDURE STOREXML2 (
   IN V_BANKSTMT XML,
   OUT V_MSG_ID VARCHAR(35),
   OUT V_CREDTTM TIMESTAMP,
   OUT V_MINISTMT XML,
   OUT ERROR_MESSAGE VARCHAR(1000) )
      LANGUAGE SQL
      MODIFIES SQL DATA
      DISABLE DEBUG MODE

BEGIN

DECLARE v_sql VARCHAR(2048) ;
DECLARE VALIDXML XML ;
DECLARE SQLCODE INTEGER ;
DECLARE SQLSTATE CHAR(5) ;
DECLARE SQLERRMC VARCHAR(70) ;
DECLARE ERROR_MESSAGE VARCHAR(250) ;
DECLARE INVALID_DOCUMENT CODITION FOR SQLSTATE '2200M'; 1

DECLARE c1 CURSOR with return to caller FOR stmt;

DECLARE EXIT HANDLER FOR INVALID_DOCUMENT
   BEGIN
   set ERROR_MESSAGE = 'DB2 DIAGNOSTICS - SQLCODE= '
      concat CHAR(SQLCODE)
      concat  ' SQLSTATE= '
      concat SQLSTATE
      concat ' SQLERRMC= '
      concat SQLERRMC ;
   insert into INVALID_DOCS ( INSERT_TIME, INSERT_DOC, ERROR_MESSAGE )
      values ( current timestamp, V_BANKSTMT, ERROR_MESSAGE ) ;
   END ; 2

SET VALIDXML = (
  DSN_XMLVALIDATE(V_BANKSTMT, 'SYSXSR.SG247915_01')) ; 3

SELECT X.MSG_ID, X.CRE_DT_TM, X_MINISTMT
   INTO V_MSG_ID, V_CREDTTM, V_MINISTMT
      FROM XMLTable(XMLNAMESPACES(
         DEFAULT 'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02'),
         '$d/Document/BkToCstmrStmt' PASSING VALIDXML as "d"
         COLUMNS
            "MSG_ID" VARCHAR(35) PATH './GrpHdr/MsgId',
            "CRE_DT_TM" TIMESTAMP PATH './GrpHdr/CreDtTm',
            "X_MINISTMT" XML PATH './Stmt' ) AS X ; 4

INSERT INTO BK_TO_CSTMR_STMT_MANUALVALIDATE
   (MSG_ID , MSG_CRE_DT_TM, BK_TO_CSTMR_STMT)
   values ( V_MSG_ID, V_CREDTTM, VALIDXML ) ;

END  !
```

The following notes explain the annotated steps in the source code for the stored procedure.

1. This statement declares an anticipated error condition for the validation of the received XML document.

2. This statement defines the processing logic when the declared error conditions are encountered.

3. This call to the DSN_XMLVALIDATE function is the call that is likely to generate the error condition that the error handling routine is setup for.

4. This statement is an XMLTABLE operation, which is more efficient than multiple XMLQUERY operations when you want to strip multiple elements from the XML document.

Another Java program (`Teststorexml2.jave`) is included in the additional materials of this book to act as a test driver for the stored procedure. See Appendix B, “Additional material” on page 277.

## 6.1.3 Stored procedures development tools

The focus of this book is the XML capabilities of DB2 for z/OS. However, also be aware of the development tools that are available for building native stored procedures with XML.

Simple native SQL procedures, such as those listed previously, can easily be coded and tested with a text editor. More complex stored procedures can benefit from the stored procedure development and debug capabilities of IBM Data Studio.

IBM Data Studio can be downloaded, at no cost, from the following address:

http://www.ibm.com/developerworks/downloads/im/data/

Data Studio provides a wide range of support for developers and DBA. Specifically, with regard to SQL stored procedures, Data Studio offers the following features:

- Supports the build, test, optimization and deployment of SQL stored procedures with an interactive routine debugger.
- Provides drag-and-drop query builders with editors for SQL and SQL/XML.
- Supports XML data type too.

For detailed documentation about using Data Studio for stored procedure development, see Part 6. “Cool tools for an easier life” of *DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond,* SG24-7604.

# 6.2 Receiving XML messages from MQ

The application scenario in Chapter 3, “Application scenario” on page 47 describes the increasingly common environment where organizations make extensive use of event-driven, messaging and workload systems. Such systems commonly send XML messages over their enterprise service bus.

In enterprises that use z/OS mainframes, WebSphere MQ commonly is the transport for XML messages. DB2 provides a range of DB2 scalar functions and table functions for working with WebSphere MQ. DB2 also provides the MQ listener, which receives messages from a message queue and calls a DB2 stored procedure with the contents of the message.

This section provides worked examples of how the ISO 20022 Bank-To-Customer-Statement message can be received from WebSphere MQ, and ingested by DB2.

## 6.2.1 WebSphere MQ functions

DB2 provides a range of built in functions for integrating with WebSphere MQ. These functions have been updated in DB2 10, so that they are based on the newer MQI interface to WebSphere MQ, rather than the older AMI interface, which was used in DB2 8. Note the following information:

- Installation job DSNTIJRT must be run to install the DB2 MQ functions.
- Installation job DSNTEJMQ verifies the MQ environment setup.

The MQ functions in DB2 10 reside in schema DB2MQ and run in WLM environment DSNWLM_MQSERIES.

To use the DB2 MQ functions within SQL statements, the MQ functions must be defined as services to DB2. These service definitions are used to encapsulate the MQ programming properties that MQ requires, so that the DB2 programmer can use a range of relatively simple functions to access WebSphere MQ. The programmer simply has to use a system-provided MQ function (such as MQSEND), in conjunction with an MQ service name.

The MQ services and policies must be defined in the following tables:

- SYSIBM.MQSERVICE_TABLE
  - Contains a list of MQ services available within a DB2 subsystem.
  - An *MQ service* is a definition of a queue, its queue manager, and code page properties of that queue.
- SYSIBM.MQPOLICY_TABLE
  - Contains a list of MQ service policies that can be used.
  - An *MQ service policy* defines MQ properties, such as message priorities, retries, exception handling that are to be used by DB2 when accessing a message queue.

We have defined an MQ service using the SQL insert in Example 6-5. We have not explicitly defined an MQ service policy, which means that we can call the DB2 MQ functions without specifying an MQ service policy, and we implicitly accept the default MQ service policy that was defined by the installation jobs.

*Example 6-5 Populating the MQSERVICE_TABLE*

```
INSERT INTO SYSIBM.MQSERVICE_TABLE (
   SERVICENAME, QUEUEMANAGER, INPUTQUEUE, CODEDCHARSETID, ENCODING, DESCRIPTION)
   VALUES ('XMLS1', 'MQBA', 'MQL.INPUT01', DEFAULT, DEFAULT, DEFAULT)
```

Actually, a more efficient way is to receive an XML document from an external source as a VARCHAR or a CLOB if you plan to validate the document against an XML schema.

The MQ scalar functions provided by DB2 are summarized in Table 6-1 on page 97.

*Table 6-1 MQ scalar functions provided by DB2 10*

| DB2 MQ function | Invocation parameters | Description |
|---|---|---|
| MQREAD | receive-service, service-policy | Reads a message as a VARCHAR (32704 bytes maximum) from a specified queue, using a specified policy, and leaves the message at the head of the queue. An empty queue returns null. |
| MQREADCLOB | receive-service, service-policy | Reads a message as a CLOB (2 MB maximum) from a specified queue, using a specified policy, and leaves the message at the head of the queue. An empty queue returns null. |
| MQRECEIVE | receive-service, service-policy, correlation-id | Reads a message (with matching correlation-id, if specified) as a VARCHAR (32704 bytes maximum) from a specified queue, using a specified policy, and removes the message from the head of the queue. An empty queue returns null. |
| MQRECEIVECLOB | receive-service, service-policy, correlation-id | Reads a message (with matching correlation-id, if specified) as a CLOB (2 MB maximum) from a specified queue, using a specified policy, and removes the message from the head of the queue. An empty queue returns null. |
| MQSEND | send-service, service-policy, msg-data, correlation-id | Writes a message (with correlation-id, if specified) as a VARCHAR (32707 bytes maximum) or CLOB (2 MB maximum) to a specified queue, using a specified policy. |

DB2 provides additional MQ table functions and MQ publishing functions as listed in *DB2 10 for z/OS Installation and Migration Guide,* GC219-2974.

Examples of the MQREAD and MQSEND functions that can be executed from SPUFI (or any other SQL editor) are shown in Example 6-6. Several invocation parameters are optional.

*Example 6-6 Sample MQREAD and MQSEND function calls*

```
select db2mq.mqread('XMLS1')
   from sysibm.sysdummy1

-- this SQL statement returns the contents of the first message on the queue
-- referenced by the XMLS1 MQ service as a string.

select db2mq.mqsend('XMLS1','<testxml><tag1>newvalue</tag1></testxml>')
   from sysibm.sysdummy1

-- this SQL statement puts the simple XML string onto the queue referenced by the
-- XMLS1 MQ service as a string.
```

No DB2 MQ functions directly use XML data type parameters. When using functions such as MQSEND and MQREAD to send XML documents to and receive them from MQ, the XML document must be passed as a string data type, and then converted to XML.

XML schema validation (where automatic or using DSN_XMLVALIDATE) accepts input only in string format. If you use XML type, DB2 must parse the XML document into internal format during the parameter passing phase, then implicitly serialize it back to string before validation, which is slower.

### 6.2.2 DB2 stored procedure reading from MQ

When you understand the MQ scalar functions that are available, you may now write a native stored procedure that reads an XML message from a message queue (as a string), stores it in a DB2 table (as an XML data type), and performs any other useful processing against the parsed XML document whilst it is in memory. The stored procedure in Example 6-7 performs the following steps:

1. Reads XML document as a string from WebSphere MQ.
2. Parses the string into an XML data type (an automatically validates that it is well formed).
3. Performs schema validation against a stored XML schema in the XML schema repository
4. Extracts specific fields from the XML document, which we want to store in relational columns in the message logging table.
5. Writes the XML document, and stripped relational fields to the message logging table.

*Example 6-7 Stored procedure to read XML message from MQ*

```
CREATE PROCEDURE STOREXML3 (
   OUT V_MSG_ID VARCHAR(35),
   OUT V_CREDTTM TIMESTAMP,
   OUT MYOUTPUT VARCHAR(1))
      LANGUAGE SQL
      MODIFIES SQL DATA
      DISABLE DEBUG MODE

BEGIN
DECLARE OUTPUTMQ CLOB ;
DECLARE VALIDXML XML ;
DECLARE MQSVC CHAR(5) ;
DECLARE SQLCODE INTEGER ;
SET MQSVC = 'XMLS1' ;

SET OUTPUTMQ = ( select db2mq.mqread(MQSVC) from sysibm.sysdummy1 ) ; 1

SET VALIDXML = (
   SELECT DSN_XMLVALIDATE(OUTPUTMQ, 'SYSXSR.SG247915_01')
   FROM SYSIBM.SYSDUMMY1 ) ; 2

SELECT X.MSG_ID, X.CRE_DT_TM
   INTO V_MSG_ID, V_CREDTTM
   FROM XMLTable(XMLNAMESPACES(
      DEFAULT 'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02'),
      '$d/Document/BkToCstmrStmt' PASSING VALIDXML as "d"
      COLUMNS
         "MSG_ID" VARCHAR(35) PATH './GrpHdr/MsgId',
         "CRE_DT_TM" TIMESTAMP PATH './GrpHdr/CreDtTm' ) AS X ; 3

INSERT INTO BK_TO_CSTMR_STMT_MANUALVALIDATE (
   MSG_ID , MSG_CRE_DT_TM, BK_TO_CSTMR_STMT)
   values ( V_MSG_ID, V_CREDTTM, VALIDXML ) ; 4

END  !
```

The numbered steps in the stored procedure are as follows:

**1.** This statement reads the XML message from MQ, and assigns it to variable OUTPUTMQ, which is defined as a CLOB.

**2.** This statement validates the CLOB that was read from WebSphere MQ against the ISO 20022 schema that was registered in DB2 XSR.

**3.** This statements strips selected data elements from the XML document.

**4.** This statement inserts the XML document and stripped elements into a relational table.

The stored procedure was written to read the first message only from the message queue, and process it. In practice, however, you modify the stored procedure to call the DB2MQ.MQRECEIVE function, so as to receive the message from the queue, and remove it within the unit of work that is controlled by the DB2 stored procedure.

### 6.2.3 DB2 MQ listener automation

With a simple programming loop, the stored procedure in 6.2.2, "DB2 stored procedure reading from MQ" on page 98 can be extended to drain the queue of all input messages. The procedure can be scheduled by some application environment to execute periodically. However, a simpler approach is to use the MQ listener, which handles all application programming and scheduling effort for you. The MQ listener provides the capability to listen to a message queue for messages when they arrive, and when they do, automatically call a stored procedure. It saves you writing and scheduling an application to poll MQ for messages and process them when they do arrive.

The MQ Listener is a standard component of DB2, which is installed with job `SDSNSAMP(DSNTIJML)`. The listener runs under UNIX System Services in z/OS. It can be invoked using UNIX System Services commands, as shown in the next examples. A JCL sample, which can invoke these commands, is in found in the `SDSNSAMP(DSNTEJML)` job.

After installing the MQ listener, it must be configured and started. The UNIX System Services command to configure the MQ listener to listen on an input queue, is shown in Example 6-8. In this example, we define a two-phase commit MQ listener process (**`db2mqln2`** command implies this) in DB2 subsystem DB0B to listen to the MQL.INPUT01 queue in Queue Manager MQBA and to call the XMLR3.STOREXML4 procedure when a message is received. The configuration is stored under the label XML1.

*Example 6-8 Command to configure MQ listener*

```
db2mqln2 add
   -ssID DB0B
   -config XML1
   -queueManager MQBA
   -inputQueue MQL.INPUT01
   -procName STOREXML4
   -procSchema XMLR3
   -numInstances 1
```

The configuration can be verified with the **show** command, as shown in Example 6-9.

*Example 6-9 Command to show MQ listener configuration*

```
db2mqln2 show -ssID DB0B -config all

returns:

   configurationName: XML1
   queueManager:      MQBA
   inputQueue:        MQL.INPUT01
   procSchema:        XMLR3
   procName:          STOREXML4
   numInstances:      1
   mqCoordinated:     T
```

The underlying DB2 table that stores the MQ listener configuration data is SYSMQL.LISTENERS, which is shown in Example 6-10.

*Example 6-10 Contents of SYSMQL.LISTENERS*

```
SYSMQL.LISTENERS

   CONFIGURATIONNAME = XML1
   QUEUEMANAGER      = MQBA
   INPUTQUEUE        = MQL.INPUT01
   PROCNODE          =
   PROCSCHEMA        = XMLR3
   PROCNAME          = STOREXML4
   PROCTYPE          = 1
   NUMINSTANCES      = 1
   WAITMILLIS        = 50
   MINQUEUEDEPTH     = 1
```

For a stored procedure to be able to work with the MQ listener, it must be coded with INPUT and OUTPUT characters that may only be VARCHAR, VARBINARY, BLOB or CLOB. The stored procedure to read an XML message from a message queue, listed in Example 6-7 on page 98, can be modified to work with the MQ listener by making a few small edits, as shown in Example 6-11.

*Example 6-11 Stored procedure modified for MQ listener integration*

```
CREATE PROCEDURE STOREXML3 (
   OUT V_MSG_ID VARCHAR(35),
   OUT V_CREDTTM TIMESTAMP,
   OUT MYOUTPUT VARCHAR(1))
     LANGUAGE SQL
     MODIFIES SQL DATA
     DISABLE DEBUG MODE

BEGIN
DECLARE OUTPUTMQ CLOB ;
DECLARE VALIDXML XML ;
DECLARE MQSVC CHAR(5) ;
DECLARE SQLCODE INTEGER ;

SET MQSVC = 'XMLS1' ;
SET OUTPUTMQ = ( select db2mq.mqread(MQSVC) from sysibm.sysdummy1 ) ;

SET VALIDXML = ( SELECT DSN_XMLVALIDATE(OUTPUTMQ, 'SYSXSR.SG247915_01')
   FROM SYSIBM.SYSDUMMY1 ) ;

SELECT X.MSG_ID, X.CRE_DT_TM INTO V_MSG_ID, V_CREDTTM
   FROM XMLTable(XMLNAMESPACES(DEFAULT
     'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02'),
      '$d/Document/BkToCstmrStmt' PASSING VALIDXML as "d"
     COLUMNS
      "MSG_ID" VARCHAR(35) PATH './GrpHdr/MsgId',
      "CRE_DT_TM" TIMESTAMP PATH './GrpHdr/CreDtTm' ) AS X ;
INSERT INTO BK_TO_CSTMR_STMT_MANUALVALIDATE (
   MSG_ID , MSG_CRE_DT_TM, BK_TO_CSTMR_STMT)
   values ( V_MSG_ID, V_CREDTTM, VALIDXML ) ;

END !
```

The listener can be operated with the commands, as shown in Example 6-12.

*Example 6-12 Commands to operate MQ listener*

```
db2mqln2 run
   -ssID DBOB
   -config XML1
   -adminQueue MQL.ADMIN
   - adminQMgr MQBA

db2mqln2 admin
   -adminQueue MQL.INPUT01
   -adminQMgr MQBA
   -adminCommand shutdown
```

The previous examples show how native stored procedures can be integrated with WebSphere MQ to receive XML messages, process them, and log them to DB2.

## 6.3 Audit queries (against logged XML messages)

Now that we have received a stream of XML messages from WebSphere MQ, and stored them in DB2, we have the ability to query them.

You do not necessarily need any special XML-enabled query tools to develop and run audit queries. The reason is because the SQL/XML language provides a range of functions to search within the XML documents for data of interest, without materializing XML structures that the query tool has to handle.

If you do have an XML-enabled query tool, SQL/XML can of course return the results of queries as XML documents. The examples that follow include a mixture of both.

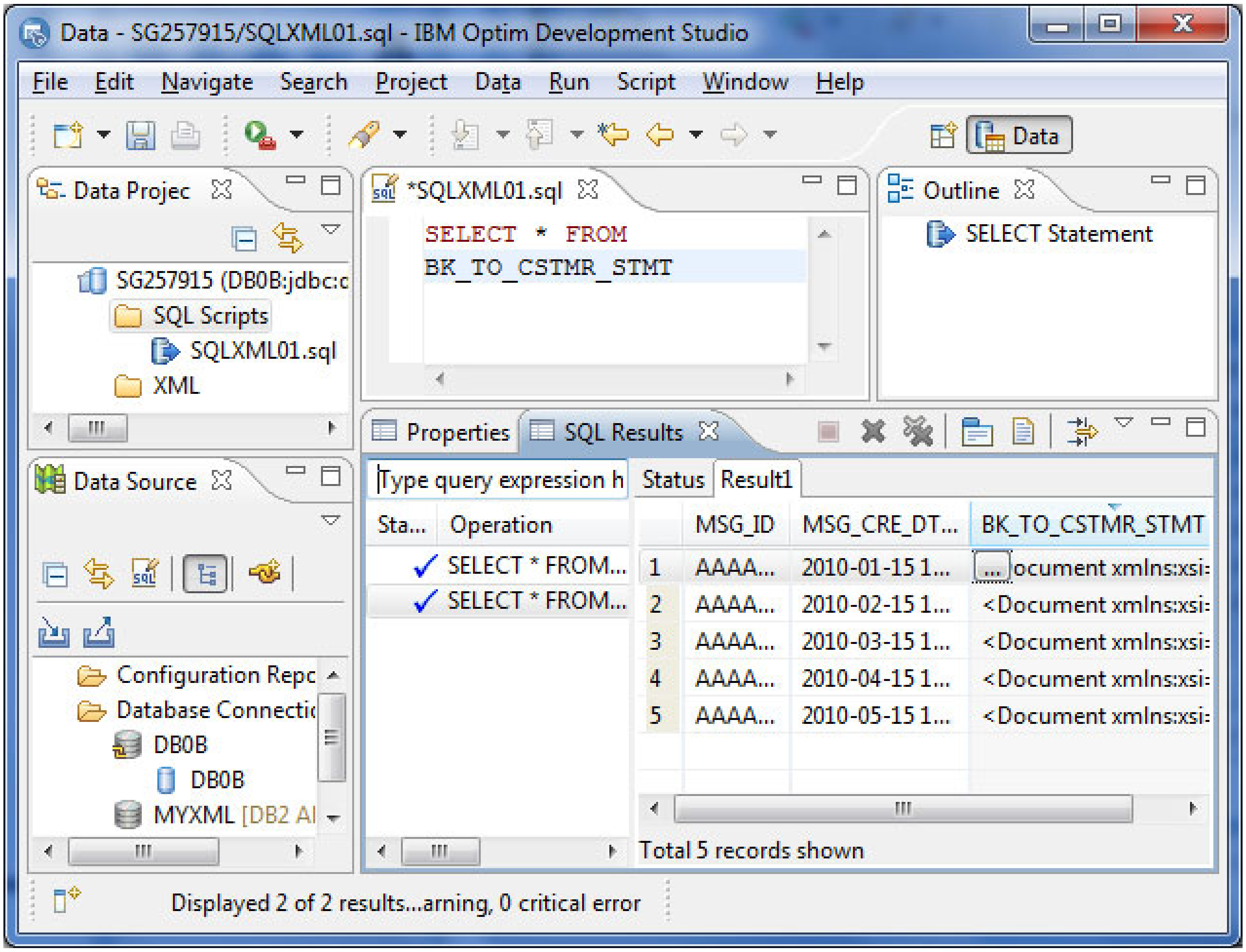
The first several examples in this section include screen captures from Optim Development Studio. This tool choice is outlined in 2.3.2, “GUI-based tools” on page 45, which provides you with an idea of the interface style of the Optim Development Studio tool.

### 6.3.1 Simple SQL/XML search examples

The SQL/XML queries in this section covers a simple case of our BK_TO_CSTMR_STMT table with five rows populated in it. The five rows represent five account statements, from January 2010 through to May 2010.

First, we look at the entire table. Figure 6-1 shows the Optim Development Studio being used to execute the following statement:

```
SELECT * FROM BK_TO_CSTMR_STMT
```



```
SELECT * FROM
BK_TO_CSTMR_STMT
```

| Sta... | Operation |
|---|---|
| ✓ | SELECT * FROM... |
| ✓ | SELECT * FROM... |

| | MSG_ID | MSG_CRE_DT... | BK_TO_CSTMR_STMT |
|---|---|---|---|
| 1 | AAAA... | 2010-01-15 1... | ... ocument xmlns:xsi: |
| 2 | AAAA... | 2010-02-15 1... | <Document xmlns:xsi: |
| 3 | AAAA... | 2010-03-15 1... | <Document xmlns:xsi: |
| 4 | AAAA... | 2010-04-15 1... | <Document xmlns:xsi: |
| 5 | AAAA... | 2010-05-15 1... | <Document xmlns:xsi: |

Total 5 records shown

Displayed 2 of 2 results...arning, 0 critical error

*Figure 6-1 SQL query: SELECT * FROM BK_TO_CSTMR_STMT*

The query was executed through Optim Development Studio, which includes the following panes:

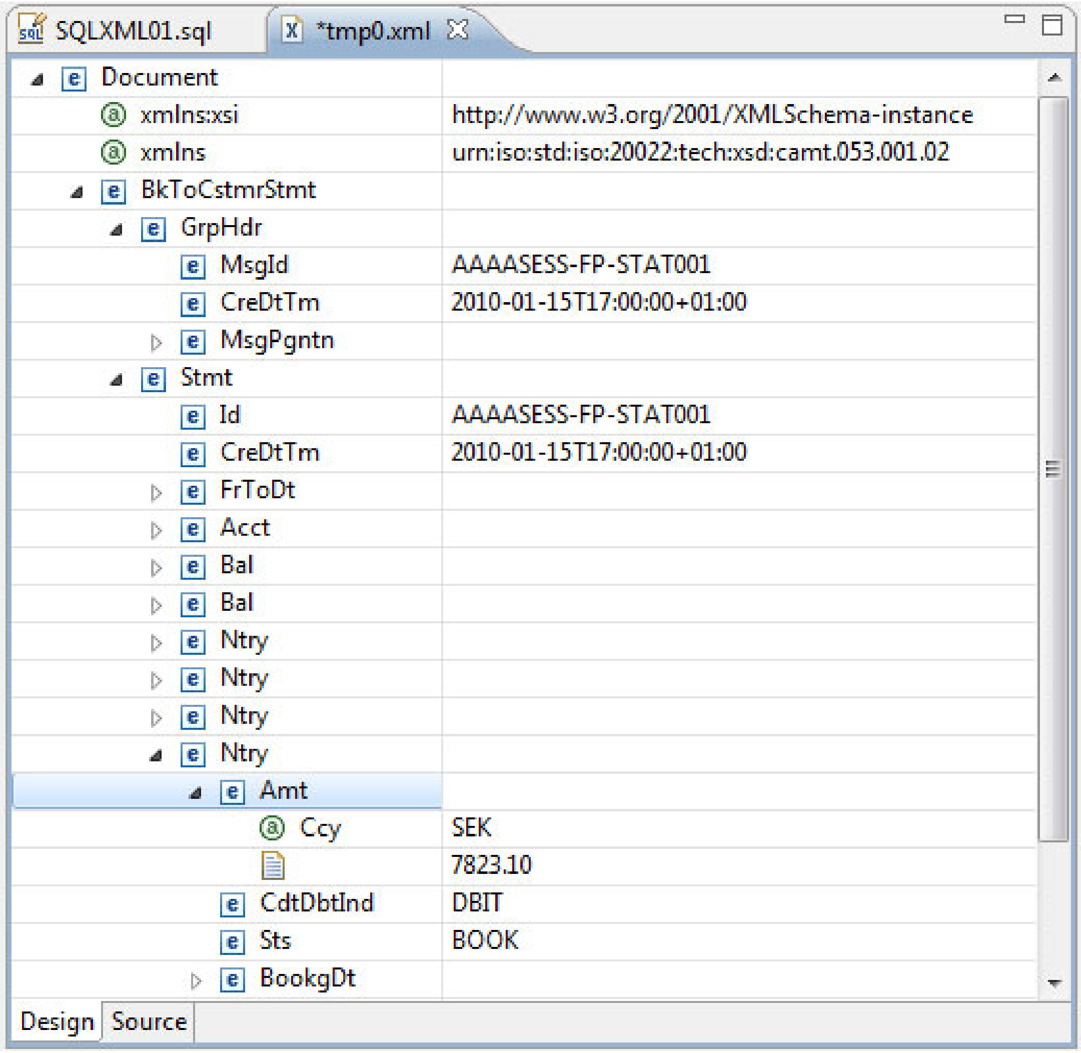
- The top pane is the project explorer, where we save our source code such as SQL statements and stored procedures.
- The bottom left pane shows database connection, in this case, to DB0B.
- The middle top pane shows the SQL statement, which we can run from the action bar.
- The bottom right pane shows the results of the SQL statement.

The table contains the XML documents that we received from MQ, and the two fields that we stripped out using XMLQUERY and XMLTABLE functions in the stored procedure examples.

To view the hidden DOCID column too, we can explicitly select the following columns:

- DB2_GENERATED_DOCID_FOR_XML
- MSG_ID
- MSG_CRE_DT_TM
- BK_TO_CSTMR_STMT

Optim Development Studio also provides an XML document viewer. Click any of the BK_TO_CSTMR_STMT XML documents in the results pane to browse the contents of the XML document, and drill down to look at specific element and attribute values at any level within the XML document. The contents of the first XML document (MSG_ID AAAASESS-FP-STAT001) are illustrated in Figure 6-2.

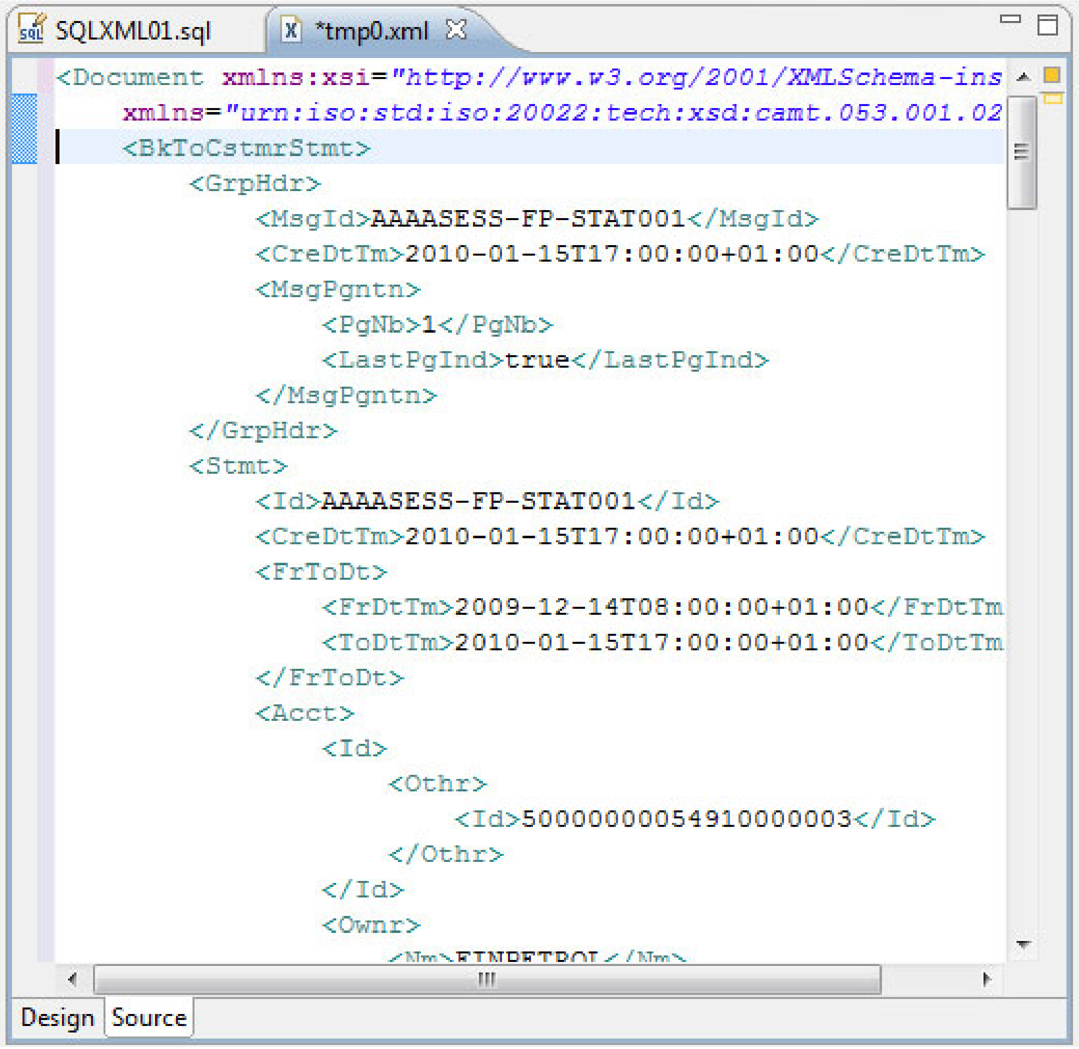


| | |
|---|---|
| Document | |
| xmlns:xsi | http://www.w3.org/2001/XMLSchema-instance |
| xmlns | urn:iso:std:iso:20022:tech:xsd:camt.053.001.02 |
| BkToCstmrStmt | |
| GrpHdr | |
| MsgId | AAAASESS-FP-STAT001 |
| CreDtTm | 2010-01-15T17:00:00+01:00 |
| MsgPgntn | |
| Stmt | |
| Id | AAAASESS-FP-STAT001 |
| CreDtTm | 2010-01-15T17:00:00+01:00 |
| FrToDt | |
| Acct | |
| Bal | |
| Bal | |
| Ntry | |
| Ntry | |
| Ntry | |
| Ntry | |
| Amt | |
| Ccy | SEK |
| | 7823.10 |
| CdtDbtInd | DBIT |
| Sts | BOOK |
| BookgDt | |

*Figure 6-2 Optim Development Studio XML document viewer: Design view*

Figure 6-2 shows that we have expanded the values of several data elements in the `GrpHdr` node. It also shows that the statement node in this particular message has four `Ntry` elements, each representing a credit or debit transaction on the account, during the period covered by this statement. We can use this viewer to drill down and examine each element in the XML document.

We can also click the **Source** tab at the bottom of the pane, and view the Source and Format to ensure that the source view is formatted for easy viewing. See Figure 6-3.



```
<Document xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
    xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02
    <BkToCstmrStmt>
        <GrpHdr>
            <MsgId>AAAASESS-FP-STAT001</MsgId>
            <CreDtTm>2010-01-15T17:00:00+01:00</CreDtTm>
            <MsgPgntn>
                <PgNb>1</PgNb>
                <LastPgInd>true</LastPgInd>
            </MsgPgntn>
        </GrpHdr>
        <Stmt>
            <Id>AAAASESS-FP-STAT001</Id>
            <CreDtTm>2010-01-15T17:00:00+01:00</CreDtTm>
            <FrToDt>
                <FrDtTm>2009-12-14T08:00:00+01:00</FrDtTm
                <ToDtTm>2010-01-15T17:00:00+01:00</ToDtTm
            </FrToDt>
            <Acct>
                <Id>
                    <Othr>
                        <Id>50000000054910000003</Id>
                    </Othr>
                </Id>
                <Ownr>
                    <Nm>FINPETROL</Nm>
```

*Figure 6-3 Optim Development Studio XML document viewer: Source view*

Conceptually, the information in this XML document is the same as what you see in your monthly account statement from your own bank, but it looks different in XML format. Therefore, how easy is it to use SQL/XML to query the document and provide a more commonly recognizable view of the data?

The XMLTABLE function is designed for transforming elements from an XML document into a tabular view, similar to your monthly bank statement. The SQL/XML query in Example 6-13 on page 105 retrieves all the debit and credit transactions from the XML document in a tabular format that looks much more like a traditional bank statement.

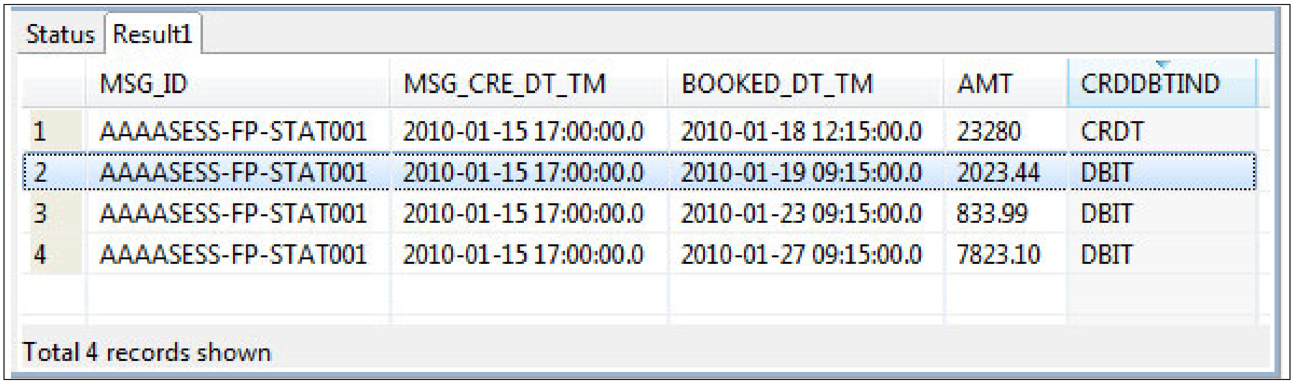
*Example 6-13 SQL/XML Query to yield a "traditional" style bank statement*

```
SELECT C.MSG_ID, C.MSG_CRE_DT_TM, X.BOOKED_DT_TM, X.AMT, X.CRDDBTIND 1
FROM xmlr3.BK_TO_CSTMR_STMT as C,
XMLTable(XMLNAMESPACES(
   DEFAULT 'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02'),2
   '$d/Document/BkToCstmrStmt/Stmt/Ntry' PASSING C.BK_TO_CSTMR_STMT as "d"
      COLUMNS 3
      "MSG_ID" VARCHAR(35) PATH '../../GrpHdr/MsgId',
      "CRE_DT_TM" TIMESTAMP PATH '../../GrpHdr/CreDtTm' ,
      "BOOKED_DT_TM" TIMESTAMP PATH './BookgDt/DtTm' ,
      "AMT" VARCHAR(50) PATH './Amt/text()' ,
      "CRDDBTIND" VARCHAR(50) PATH './CdtDbtInd/text()' ) AS X
   where C.MSG_ID = 'AAAASESS-FP-STAT001' 4
   order by X.BOOKED_DT_TM asc ; 5
```

The highlighted numbers of the SQL/XML query in Example 6-13 are explained as follows:

1. We select three data elements from the XML document, which are the transaction timestamp, transaction amount, and credit/debit indicator of each transactional entry in the BK_TO_CSTMR_STMT XML document. Additionally, we take the statement message_id and statement date from the base table.
2. As always, we must specify the correct namespaces in the XQuery expression.
3. Important: We must specify the node within the XML document on which the XMLTABLE function is operating. Usually this is at the node where we want to extract the detailed information. In this case, we base it on the Ntry node, because we want to retrieve a separate row in the result set for each transactional entry in the XML document.
4. Because we are focussing this query on a single XML document, we use a predicate on the base table column to return data from only the document with MSG_ID of AAAASESS-FP-STAT0001.
5. Finally, we can order the result set by using either a relational column or an XML data element. In this case, we want to order the result set by the booked transaction date of each Ntry node in the XML document.

The results are displayed in Figure 6-4. Note that all returned rows contain the same values for columns from the base table (relating to the bank statement XML document) but unique values for the columns relating to individual entries from the document.



| | MSG_ID | MSG_CRE_DT_TM | BOOKED_DT_TM | AMT | CRDDBTIND |
|---|---|---|---|---|---|
| 1 | AAAASESS-FP-STAT001 | 2010-01-15 17:00:00.0 | 2010-01-18 12:15:00.0 | 23280 | CRDT |
| 2 | AAAASESS-FP-STAT001 | 2010-01-15 17:00:00.0 | 2010-01-19 09:15:00.0 | 2023.44 | DBIT |
| 3 | AAAASESS-FP-STAT001 | 2010-01-15 17:00:00.0 | 2010-01-23 09:15:00.0 | 833.99 | DBIT |
| 4 | AAAASESS-FP-STAT001 | 2010-01-15 17:00:00.0 | 2010-01-27 09:15:00.0 | 7823.10 | DBIT |

Total 4 records shown

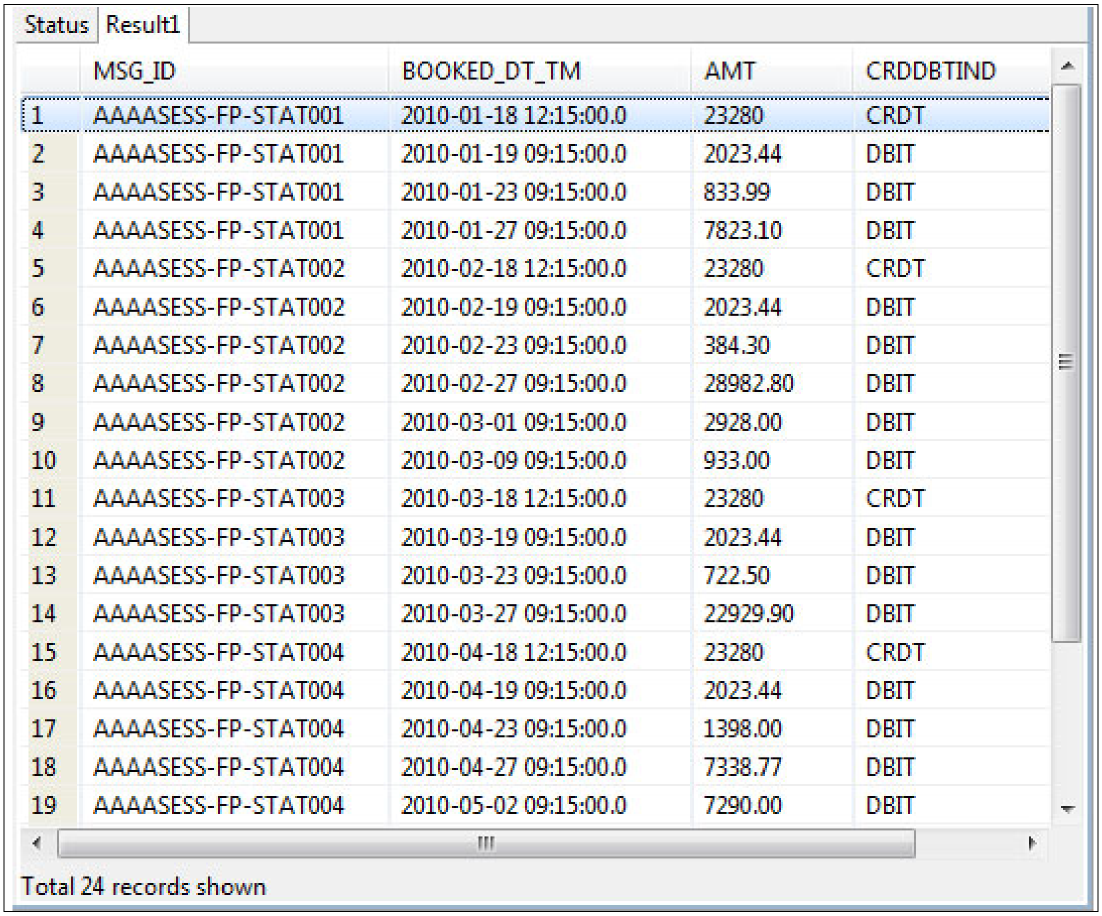
*Figure 6-4 Tabular result set of bank statement entries*

The contents of XML data from multiple documents can be returned in a single relational result set, as show by the SQL/XML statement in Example 6-14 on page 106, which retrieves all five bank statements (January - May) and produces a consolidated bank statement for the entire period.

*Example 6-14 SQL/XML query spanning multiple XML documents*

```
SELECT C.MSG_ID, X.BOOKED_DT_TM, X.AMT, X.CRDDBTIND
   FROM xmlr3.BK_TO_CSTMR_STMT as C,
   XMLTable(XMLNAMESPACES(
      DEFAULT 'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02'),
      '$d/Document/BkToCstmrStmt/Stmt/Ntry' PASSING C.BK_TO_CSTMR_STMT as "d"
      COLUMNS
         "MSG_ID" VARCHAR(35) PATH '../../GrpHdr/MsgId' ,
         "CRE_DT_TM" TIMESTAMP PATH '../../GrpHdr/CreDtTm' ,
         "BOOKED_DT_TM" TIMESTAMP PATH './BookgDt/DtTm' ,
         "AMT" VARCHAR(50) PATH './Amt' ,
         "CRDDBTIND" VARCHAR(50) PATH './CdtDbtInd'
) AS X
order by X.BOOKED_DT_TM asc ;
```

The results of the query in Example 6-14 are shown in Figure 6-5.



| | MSG_ID | BOOKED_DT_TM | AMT | CRDDBTIND |
|---|---|---|---|---|
| 1 | AAAASESS-FP-STAT001 | 2010-01-18 12:15:00.0 | 23280 | CRDT |
| 2 | AAAASESS-FP-STAT001 | 2010-01-19 09:15:00.0 | 2023.44 | DBIT |
| 3 | AAAASESS-FP-STAT001 | 2010-01-23 09:15:00.0 | 833.99 | DBIT |
| 4 | AAAASESS-FP-STAT001 | 2010-01-27 09:15:00.0 | 7823.10 | DBIT |
| 5 | AAAASESS-FP-STAT002 | 2010-02-18 12:15:00.0 | 23280 | CRDT |
| 6 | AAAASESS-FP-STAT002 | 2010-02-19 09:15:00.0 | 2023.44 | DBIT |
| 7 | AAAASESS-FP-STAT002 | 2010-02-23 09:15:00.0 | 384.30 | DBIT |
| 8 | AAAASESS-FP-STAT002 | 2010-02-27 09:15:00.0 | 28982.80 | DBIT |
| 9 | AAAASESS-FP-STAT002 | 2010-03-01 09:15:00.0 | 2928.00 | DBIT |
| 10 | AAAASESS-FP-STAT002 | 2010-03-09 09:15:00.0 | 933.00 | DBIT |
| 11 | AAAASESS-FP-STAT003 | 2010-03-18 12:15:00.0 | 23280 | CRDT |
| 12 | AAAASESS-FP-STAT003 | 2010-03-19 09:15:00.0 | 2023.44 | DBIT |
| 13 | AAAASESS-FP-STAT003 | 2010-03-23 09:15:00.0 | 722.50 | DBIT |
| 14 | AAAASESS-FP-STAT003 | 2010-03-27 09:15:00.0 | 22929.90 | DBIT |
| 15 | AAAASESS-FP-STAT004 | 2010-04-18 12:15:00.0 | 23280 | CRDT |
| 16 | AAAASESS-FP-STAT004 | 2010-04-19 09:15:00.0 | 2023.44 | DBIT |
| 17 | AAAASESS-FP-STAT004 | 2010-04-23 09:15:00.0 | 1398.00 | DBIT |
| 18 | AAAASESS-FP-STAT004 | 2010-04-27 09:15:00.0 | 7338.77 | DBIT |
| 19 | AAAASESS-FP-STAT004 | 2010-05-02 09:15:00.0 | 7290.00 | DBIT |

Total 24 records shown

*Figure 6-5 Relational result set spanning data elements from multiple XML documents*

A wider range of SQL/XML query techniques that use all the DB2 XML functions is described in 6.4, “SQL/XML query techniques” on page 109.

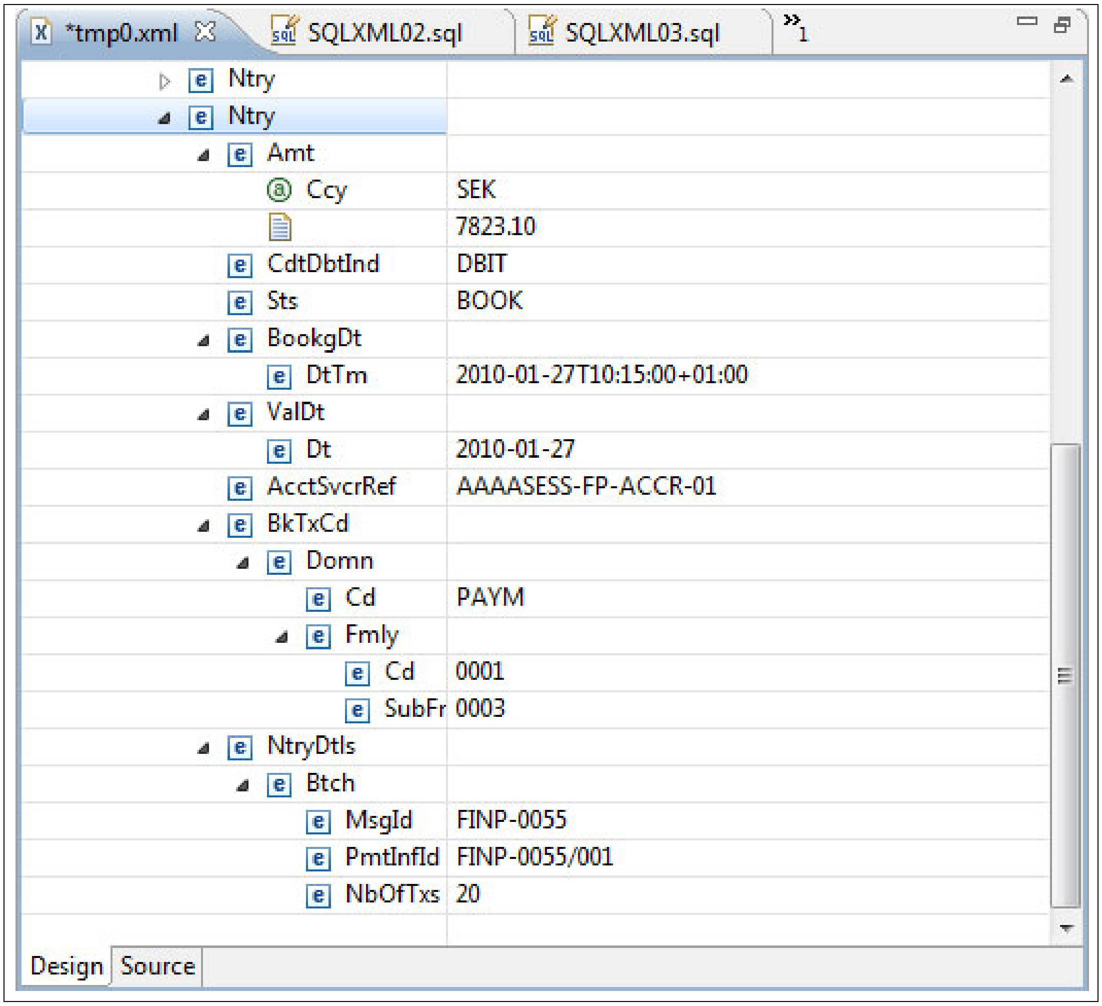
## 6.3.2 Choosing XML indexes

With a total of five rows in our table, indexes are not necessary. However, if we were storing a large number of XML documents in our XML auditing system, we would have to create XML indexes to support efficient searching within the XML documents.

As the XML audit database grows in size, the need for XML indexes becomes clear. Each SQL/XML query in 6.3.1, “Simple SQL/XML search examples” on page 102 is considered, and candidate XML indexes are identified. Chapter 11, “Performance considerations” on page 247 covers XML index design.

We consider XML index design for the specific purpose of enabling efficient search with an audit database of bank statements.

We assume that a primary purpose of the audit database is to analyze data at the individual banking transaction level. That means that the focus of many queries will be the `Ntry` nodes in these XML documents. Figure 6-6 shows the typical data elements within an `Ntry` node.



```
Ntry
Ntry
  Amt
    Ccy            SEK
                   7823.10
  CdtDbtInd        DBIT
  Sts              BOOK
  BookgDt
    DtTm           2010-01-27T10:15:00+01:00
  ValDt
    Dt             2010-01-27
  AcctSvcrRef      AAAASESS-FP-ACCR-01
  BkTxCd
    Domn
      Cd           PAYM
      Fmly
        Cd         0001
        SubFr      0003
  NtryDtls
    Btch
      MsgId        FINP-0055
      PmtInfId     FINP-0055/001
      NbOfTxs      20
```

*Figure 6-6 Typical “Ntry” node within a Bk_To_Cstmr_Stmt document*

If a decision was made that many audit queries were to focus on AccountServicerReference, we check the ISO 20022 documentation to understand the following constraints on this data element:

- AccountServicerReference
- XPath = /Document/BkToCstmrStmt/Stmt/Ntry/AcctSvcrRef
- Presence: [0..1]
- Definition: Unique reference as assigned by the account servicing institution to unambiguously identify the entry.
- Data Type: Max35Text
- Format: maxLength: 35
- The minLength: 1

There may be many occurrences of AcctSvcrRef in each Bk_To_Cstmr_Stmt document. An XML index on `/Document/BkToCstmrStmt/Stmt/Ntry/AcctSvcrRef` location will therefore have multiple entries per document. A candidate index definition is shown in Example 6-15.

*Example 6-15 Candidate XML index definitions*

```
Create index AcctSvcrRef_IX
   ON BK_TO_CSTMR_STMT(BK_TO_CSTMR_STMT)
   Generate Key using XMLPATTERN 'declare default element namespace
   "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      /Document/BkToCstmrStmt/Stmt/Ntry/AcctSvcrRef'
as SQL VARCHAR(35)  ;
```

## 6.3.3 Verifying XML index usage

Accurate data typing is critical for index eligibility. If the actual contents of the indexed data element in the document does not match the data type that specified in the XMLPATTERN of the create index statement, the index will not contain an entry for that data element.

The corollary of the previous statement is that the XML schemas must place data type constraints on fields that might be indexed, and the XML documents are validated against their schema definition. This way is the best for ensuring that effective indexes can be defined on XML columns.

The simplest way of checking that an Index will be effective is to run RUNSTATS and check the values of FIRSTKEYCARDF and FULLKEYCARDF in SYSIBM.SYSINDEXES. If you know the number of rows in the table, and you have a reasonable expectation of the average number of XML index hits per XML document, you will have an approximate idea of what the value of FULLKEYCARDF should be. For example, if you have one million rows in the BK_TO_CSTMR_STMT table, and an average of 20 `Ntry` nodes per XML document, you might be expecting FULLKEYCARDF to be about 20 million. Of course it may be less if the same values of AcctSvcrRef appear many times, but at least you can make a judgement on whether the Index has found approximately the correct number of data elements to index.

If an XML index has cardinality of 0, determine why the index did not find any matching data elements to index. The most likely reasons are a typographical error in the XMLPATTERN, or the namespace definition, or a data type mismatch.

After you determine that the index has approximately the correct number of entries in it, you must test your application SQL to determine whether the optimizer will use the XML index.

This test is a simple case of using your favorite EXPLAIN tool, and checking whether the XML indexes are being used in the access path for the query.

## 6.4 SQL/XML query techniques

Before considering more query techniques, let us review the productivity that pureXML can provide.

The extent of the work that we have done so far is as follows:

1. Download a publicly available XML schema and register it in DB2.
2. Create a simple DB2 table (with automatic schema validation) to store the corresponding XML documents that we receive.
3. *Load* the XML documents into the table (we use the term load loosely in this step, to account for many possible ways to ingest the XML documents).
4. Start writing SQL/XML queries against a rich XML data model.
5. Create an iXML index to support those queries.

We did not need to extract XML data elements from the XML documents to build relational searching columns. The only reason we chose to use the DB2 XML functions to extract selected data elements is to show the ease of moving data between XML and relational.

The only difficult tasks we have had to do is to understand the concepts of the XML model of data, and the SQL/XML extensions to the SQL language.

This section provides some sample SQL/XML audit queries to illustrate the various functions and techniques for querying XML data and how to combine them.

### 6.4.1 Manipulating XML data with XPath functions

DB2 offers a wide range of functions for use in XPath expressions, and are a subset of the XPath 2.0 standard. Use them to work directly on the data in the XPath expression instead of first having to extract the data from the document, and then manipulate or query it after.

The functions are listed in Table 6-2.

*Table 6-2 Functions and examples*

| Function | Example |
|---|---|
| Accessory functions | fn:data |
| Functions on numeric values | fn:abs |
| Functions on strings | fn:substring |
| Functions on boolean values | fn:not |
| Functions on durations, dates and times | fn:month-from-date |
| Functions on sequences | fn:distinct-values |
| Aggregate functions | fn:avg |
| Context functions | fn:position |

These functions all belong to the following namespace and have the fn default namespace prefix:

```
http://www.w3.org/2005/xpath-functions
```

Additionally, constructor functions for each XPath data type are available. Examples of these are xs:string and xs:date. The namespace for these functions is as follows and xs is the prefix:

```
http://www.w3.org/2001/XMLSchema
```

For more information about the available XPath functions, see the following address:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2z10.doc.xml/db2z_xpxqfunctionreference.htm

If you want to calculate the sum of all entries in a bank statement, you may use the fn:sum function as shown in Example 6-16.

*Example 6-16 Calculating the sum of the entries in a BankToCustomerStatement*

```
SELECT
XMLCAST ( XMLQUERY (
'declare default element namespace
"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
fn:sum(/Document/BkToCstmrStmt/Stmt/Ntry/Amt)'
PASSING BK_TO_CSTMR_STMT
) AS DECIMAL(12,2)
FROM BK_TO_CSTMR_STMT
```

The result of the XMLQUERY call is cast as DECIMAL(12,2) by using the XMLCAST function. Otherwise, the result of the query would have had type XML, although we know that the content is really numeric. To process the data further by using arithmetic functions, comparisons, or simply return it as a numeric value, we need to perform this conversion.

The use of the XMLCAST function requires that the input is a sequence of one item. If the result of the XMLQUERY expression is a longer sequence or an empty sequence, the XMLCAST function returns an error, SQL code -16003. In this example, we know that the sequence has exactly one element, because the fn:sum function is applied to all the Amt elements so it cannot be longer than one, and the result of applying fn:sum to an empty sequence is 0 (zero), so it always returns a result.

The XMLCAST function can also be used to strip any element tags if the result is a simple element.

In another example of applying XPath functions, we consider several available date functions. Date and time functionality has been included in DB2 XPath with version 10 because the functionality is an important aspect of most business applications.

You might want to select all the entries from a BankToCustomerStatement that were made in the last month. To create an XPath expression that does that, use the subtraction function for datetime, which is written as a minus sign (-), and the two constructor functions xs:dateTime and xs:yearMonthDuration. The result is shown in Example 6-17 on page 111.

*Example 6-17 Using time and date functions in an XPath expression*

```
SELECT
  XMLQUERY (
  'declare default element namespace
  "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
  Document/BkToCstmrStmt/Stmt/Ntry
  [BookgDt/DtTm>xs:dateTime($tm)-xs:yearMonthDuration("P1M")]'
  PASSING BK_TO_CSTMR_STMT, CURRENT TIMESTAMP AS "tm"
  )
FROM BK_TO_CSTMR_STMT
```

The predicate says that the DtTm element must be greater than current timestamp minus one month, where the current timestamp is given as a parameter in the passing clause, and where both this parameter and the one month constant is created using constructor functions. The predicate is applied to each `Ntry` element in the document, and when evaluated to true, the `Ntry` element will be included in the result. The reason is because even though we reference the elements BookgDt and DtTm, these are only part of the predicate and not of the path that we have selected.

## 6.4.2 Filtering the rows returned with XMLEXISTS

The query shown in Example 6-17 returns one row for each original table row, regardless of whether any entries in the BankToCustomerStatement qualified or not. For those BankToCustomerStatements that did not contain an entry less than a month old, the result of the query is the empty sequence. The reason is because predicates that are used in an XMLQUERY expression are used to filter the data returned from the expression, not to determine whether data is returned at all.

To return only the rows that actually has some contents, we can apply the same filtering predicate in the where clause through the function XMLEXISTS. This function returns false for an empty sequence and true for everything else. It has the same syntax as XMLQUERY.

The resulting query is shown in Example 6-18.

*Example 6-18 Avoiding empty sequences in result by using XMLEXISTS*

```
SELECT
  XMLQUERY (
  'declare default element namespace
  "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
  Document/BkToCstmrStmt/Stmt/Ntry
  [BookgDt/DtTm>xs:dateTime($tm)-xs:yearMonthDuration("P1M")]'
  PASSING BK_TO_CSTMR_STMT, CURRENT TIMESTAMP AS "tm"
  )
FROM BK_TO_CSTMR_STMT
WHERE XMLEXISTS (
  'declare default element namespace
  "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
  Document/BkToCstmrStmt/Stmt/Ntry
  [BookgDt/DtTm>xs:dateTime($tm)-xs:yearMonthDuration("P1M")]'
  PASSING BK_TO_CSTMR_STMT, CURRENT TIMESTAMP AS "tm"
)
```

XMLEXISTS predicates are candidates for index access if the XPath expression matches the pattern of an XML index.

> **Note:** A predicate must always be in square brackets. If the brackets are omitted, the result of the XPath expression is either *true* or *false*.
>
> Both true and false are non-empty sequences and can cause the XMLEXISTS expression to evaluate to true, thus returning all rows in the table.

## 6.4.3 Creating documents with publishing functions

In Example 6-18 on page 111 the result is a sequence of Ntry elements for each resulting row rather than an XML document. If we want to create an XML document from the result, we can use the XML publishing functionsXMLELEMENT and XMLDOCUMENT.

Example 6-19 shows that XMLELEMENT is used to create an outermost element with name `results`, and XMLDOCUMENT to create a document from this element.

*Example 6-19 Combining XMLQUERY with publishing functions*

```
SELECT
XMLDOCUMENT( XMLELEMENT(NAME "results",
  XMLQUERY (
  'declare default element namespace
  "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
  Document/BkToCstmrStmt/Stmt/Ntry
  [BookgDt/DtTm>xs:dateTime($tm)-xs:yearMonthDuration("P1M")]'
  PASSING BK_TO_CSTMR_STMT, CURRENT TIMESTAMP AS "tm"
  )
))
FROM BK_TO_CSTMR_STMT
WHERE XMLEXISTS (
  'declare default element namespace
  "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
  Document/BkToCstmrStmt/Stmt/Ntry
  [BookgDt/DtTm>xs:dateTime($tm)-xs:yearMonthDuration("P1M")]'
  PASSING BK_TO_CSTMR_STMT, CURRENT TIMESTAMP AS "tm"
)
```

The result from this query is an XML document for each Bank To Customer Statement that had any entries the last month, containing these entries.

For more information about the use of publishing functions, go to the following addresses:

- http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2z10.doc.xml/db2z_publishfuncs.htm
- http://www.ibm.com/developerworks/data/library/techarticle/dm-0511melnyk/

## 6.4.4 Aggregating documents with XMLAGG

Instead of having individual XML documents for each BankToCustomerStatement with entries from the last month, we want all these entries to be collected in one XML document regardless of their origin.

In this case, we must be able to create new XML elements *across* existing XML documents, and that is exactly what is offered by the XMLAGG publishing function. This function can take any number of XML values and create a sequence from them. It is an aggregate function in the same manner as the SQL functions AVG and MIN, so it is applied to all values from all rows in the select statement.

Example 6-20 shows how to combine Example 6-19 on page 112 with the XMLAGG function to collect all the entries in one XML document.

*Example 6-20 Using XMLAGG to consolidate all entries into one document*

```
SELECT
XMLDOCUMENT(
XMLAGG (
XMLELEMENT(NAME "results",
XMLQUERY (
'declare default element namespace
"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
Document/BkToCstmrStmt/Stmt/Ntry
[BookgDt/DtTm>xs:dateTime($tm)-xs:yearMonthDuration("P1M")]'
PASSING BK_TO_CSTMR_STMT, CURRENT TIMESTAMP AS "tm"
)
)))
FROM BK_TO_CSTMR_STMT
WHERE XMLEXISTS (
'declare default element namespace
"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
Document/BkToCstmrStmt/Stmt/Ntry
[BookgDt/DtTm>xs:dateTime($tm)-xs:yearMonthDuration("P1M")]'
PASSING BK_TO_CSTMR_STMT, CURRENT TIMESTAMP AS "tm"
)
```

Although we no longer get any rows that contain empty sequences because all the Ntry elements are collected into one document, we still include the XMLEXISTS predicate to allow for index access.

## 6.4.5 Enumerating all occurrences using XMLTABLE

Now imagine that instead of creating one XML document with all the newest entries, we instead want one XML document per entry. The function XMLTABLE can help to provide this.

The XMLTABLE function takes the following input:

- An optional namespace declaration
- A row XPath expression
- A number of column XPath expressions
- Optionally one or more input arguments as XPath variables

The row XPath expression returns a sequence of items, each producing a row in the result table of the XMLTABLE function. If the row XPath expression points to an element, the number of elements within a document with that particular XPath, determines the number of resulting rows. This way is what allows us to create one row per `Ntry` element.

Using the row XPath expression as a starting point, one or more column XPath expressions define the contents of the columns that are returned from the XMLTABLE function. Associated with each is a data type and a name that can be used in the surrounding select statement.

Example 6-21 shows how to use XMLTABLE to extract all the entries no more than a month old, returning one row per entry. We use only one XPath column expression (`'.'`), which effectively returns the contents of the row XPath expression. This Ntry element is returned with data type XML and subsequently made into an XML document by using the XMLDOCUMENT publishing function.

*Example 6-21 Extracting one entry per row using XMLTABLE*

```
SELECT XMLDOCUMENT (X.NTRY)
FROM BK_TO_CSTMR_STMT S,
XMLTABLE (XMLNAMESPACES(DEFAULT
  'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02',
  '/Document/BkToCstmrStmt/Stmt/Ntry
[BookgDt/DtTm>xs:dateTime($tm)-xs:yearMonthDuration("P1M")]'
PASSING S.BK_TO_CSTMR_STMT, current timestamp as "tm"
COLUMNS
"NTRY"      XML    PATH '.'
) X
```

We have now abandoned the XMLEXISTS predicate in the `WHERE` clause. The row XPath expressions of an XMLTABLE function is a candidate for index access, so the XMLEXISTS predicate can yield exactly the same result and is no longer needed.

More information and examples about using the XMLTABLE function is at the following address:

http://www.ibm.com/developerworks/data/library/techarticle/dm-0708nicola/

## 6.4.6 Grouping data with XMLTABLE

Finally, we consider a case where we want to return all the entries ordered into documents according to the currency in which the entry was made. Therefore, we want one document with all the entries made in U.S. dollars (USD), one with all the entries made in Swedish krona (SEK), and so on.

For each of the resulting documents we need entries from several of the original documents, but assuming that it is possible to make entries in different currencies to the same account, we cannot be sure that all entries from one account or indeed one BankToCustomerStatement will go into the same result document.

This can be obtained by combining the XMLTABLE function from Example 6-21 with column grouping, XMLAGG and publishing functions XMLELEMENT and XMLDOCUMENT. The query is shown in Example 6-22 on page 115.

We have added another column XPath expression to obtain the currency from the entry, and the result is grouped according to this currency. We then apply the XMLAGG function to the resulting entries, and wrap these in an element named Result, which is then given as input to the XMLDOCUMENT function to produce an XML document.

*Example 6-22 Grouping entries obtained from XMLTABLE according to currency*

```
SELECT XMLDOCUMENT (
XMLELEMENT(NAME "Result",
XMLAGG(X.NTRY)))
FROM BK_TO_CSTMR_STMT S,
XMLTABLE (XMLNAMESPACES(DEFAULT
  'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02',
'/Document/BkToCstmrStmt/Stmt/Ntry
[BookgDt/DtTm>xs:dateTime($tm)-xs:yearMonthDuration("P1M")]'
PASSING S.BK_TO_CSTMR_STMT, current timestamp as "tm"
COLUMNS
"CCY"       VARCHAR(20)  PATH 'Amt/@Ccy' ,
"NTRY"      XML          PATH '.'
) X
GROUP BY X.CCY
```

# 6.5 User-defined functions with XML

User-defined functions (UDFs) are a helpful way of encapsulating complex logic or expressions in functions and can be easy for anyone to use. The first stored procedure in this chapter (which received an XML document, extracted a number of relational and XML objects from the received XML document, and stored the results in a DB2 table) contained several XQuery expressions that can be challenging to use for an SQL programmer who is unfamiliar with XML.

## 6.5.1 UDFs for reading from XML documents

We can dramatically simplify that stored procedure by providing three UDFs to perform the XML tasks. Example 6-23 provides the DDL necessary to create the three UDFs that can help the traditional SQL programmer.

*Example 6-23 Creating three user-defined functions*

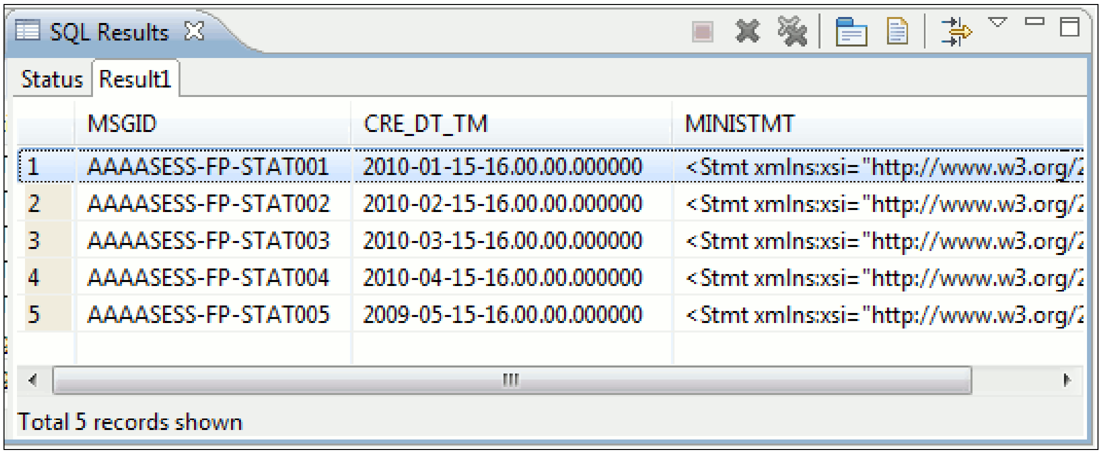
```
CREATE FUNCTION GETMSGID(doc XML) RETURNS VARCHAR(35)
   LANGUAGE SQL
   CONTAINS SQL
   NO EXTERNAL ACTION
   DETERMINISTIC
   DISABLE DEBUG MODE
BEGIN
RETURN xmlcast(xmlquery(
      'declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $d/Document/BkToCstmrStmt/GrpHdr/MsgId' passing doc as "d")
   as varchar(35) ) ;
END !

CREATE FUNCTION GETCREDTTM(doc XML) RETURNS TIMESTAMP
   LANGUAGE SQL
   CONTAINS SQL
   NO EXTERNAL ACTION
   DETERMINISTIC
   DISABLE DEBUG MODE
BEGIN
RETURN xmlcast(xmlquery(
      'declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $d/Document/BkToCstmrStmt/GrpHdr/CreDtTm' passing doc as "d")
   as timestamp ) ;
END !

CREATE FUNCTION GETMINISTMT(doc XML) RETURNS XML
   LANGUAGE SQL
   CONTAINS SQL
   NO EXTERNAL ACTION
   DETERMINISTIC
   DISABLE DEBUG MODE
BEGIN
RETURN xmlquery(
   'declare default element namespace
   "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
   $d/Document/BkToCstmrStmt/Stmt' passing doc as "d") ;
END !
```

Each of the three UDFs encapsulates XQuery functions to return a separate data type to the SQL programmer. Example 6-24 shows how these functions can be used in standard SQL.

*Example 6-24 Usage of UDFs*

```
select
   getmsgid(BK_TO_CSTMR_STMT) as MSGID,
   getcredttm(BK_TO_CSTMR_STMT) as CRE_DT_TM,
   getministmt(BK_TO_CSTMR_STMT) as MINISTMT
   from BK_TO_CSTMR_STMT
```

The results of the UDF-based query in Example 6-24 are shown in Figure 6-7.



| | MSGID | CRE_DT_TM | MINISTMT |
|---|---|---|---|
| 1 | AAAASESS-FP-STAT001 | 2010-01-15-16.00.00.000000 | <Stmt xmlns:xsi="http://www.w3.org/2 |
| 2 | AAAASESS-FP-STAT002 | 2010-02-15-16.00.00.000000 | <Stmt xmlns:xsi="http://www.w3.org/2 |
| 3 | AAAASESS-FP-STAT003 | 2010-03-15-16.00.00.000000 | <Stmt xmlns:xsi="http://www.w3.org/2 |
| 4 | AAAASESS-FP-STAT004 | 2010-04-15-16.00.00.000000 | <Stmt xmlns:xsi="http://www.w3.org/2 |
| 5 | AAAASESS-FP-STAT005 | 2009-05-15-16.00.00.000000 | <Stmt xmlns:xsi="http://www.w3.org/2 |

Total 5 records shown

*Figure 6-7 SQL results using UDFs on XML documents*

Therefore, if we rewrite the first stored procedure to use these functions (as shown in Example 6-25), it now looks much simpler. In fact, the traditional SQL programmer does not need to know any XQuery, if the programmer has a range of system-provided and user-defined functions to operate on XML data.

*Example 6-25 Modified stored procedure using UDFs instead of XQuery expressions*

```
CREATE PROCEDURE STOREXML5 (
   IN V_BANKSTMT XML,
   OUT V_MSG_ID VARCHAR(35),
   OUT V_CREDTTM TIMESTAMP,
   OUT V_MINISTMT XML)
      LANGUAGE SQL
      MODIFIES SQL DATA
      DISABLE DEBUG MODE

BEGIN

DECLARE VALIDXML XML ;
DECLARE SQLCODE INTEGER ;

SET VALIDXML = (
   SELECT DSN_XMLVALIDATE(V_BANKSTMT, 'SYSXSR.CAMT_053_001_02');

SELECT
   getmsgid(BK_TO_CSTMR_STMT),
   getcredttm(BK_TO_CSTMR_STMT),
   getministmt(BK_TO_CSTMR_STMT)
   into V_MSG_ID, V_CREDTTM, V_MINISTMT from BK_TO_CSTMR_STMT;

INSERT INTO BK_TO_CSTMR_STMT_MANUALVALIDATE (
   MSG_ID , MSG_CRE_DT_TM, BK_TO_CSTMR_STMT)
   values ( V_MSG_ID, V_CREDTTM, VALIDXML ) ;

END  !
```

UDFs for XML are a great way to take advantage of DB2's pureXML capabilities, without necessarily exposing the SQL programmers to the full complexity of XQuery expressions.

### 6.5.2 UDFs for writing updates to XML documents

The UDF examples we have shown so far have been based around retrieving data from XML documents, and transforming them to traditional relational objects. UDFs can also be used for other purposes, such as writing sub-document updates.

Example 6-26 on page 118 illustrates a UDF that inserts a new XML node into an XML document.

*Example 6-26 UDF for XML sub-document update*

```
CREATE FUNCTION UPDATE_ADDR(NEW_ADDR CLOB)
   RETURNS integer
   LANGUAGE SQL
   MODIFIES SQL DATA
   DETERMINISTIC
   DISABLE DEBUG MODE
BEGIN
UPDATE CUSTOMER_TABLE
   SET CUSTOMER_ADDRESS = XMLMODIFY(
      'insert node $new as last into /Customer/addresses',
      XMLPARSE(NEW_ADDR) as "new" ) ;
   return SQLCODE;
END !

-- This UDF could be called using the following SQL statement -

SELECT UPDATE_ADDR(V_ADDR) INTO UDF_RETURN1
   FROM CUSTOMER_TABLE
   where CUSTID = ‘VALUE’ ;
```

# 6.6 Triggers with XML

Triggers are a popular way of implementing database dependencies. Triggers can be defined to perform procedural work as a direct result of an SQL write (insert, update or delete) to the table on which the trigger is defined. Triggers can use *transition variables* to reference the new or old values of DB2 columns, enabling the trigger routine body to work with the data values that have just been written.

Triggers can be used on tables with XML columns. However, the new and old values of XML columns are not available as transition variables to the trigger routine body.

In summary, you can continue to write triggers with tables that have XML columns. However, if you want to reference the contents of an XML column, you must reread the XML data by using one of the relational columns that are available as a transition variable.

# 6.7 XML joins

In this section, we examine the 6.7.1, “XML to relational join” on page 118, and the 6.7.2, “XML to XML join” on page 120.

## 6.7.1 XML to relational join

Joining XML data with relational data is no different in principle to wholly relational joins. The data types of the join must be compatible, and indexes should be used to make the join perform well.

The ISO 20022 sample XML messages that we are using all relate to an account called FINPETROL. The account name is stored at the following XPath location:

```
/Document/BkToCstmrStmt/Stmt/Acct/Ownr/Nm
```

We can perform a join based on the account name within the XML document, against a relational table of addresses, illustrated in Example 6-27.

*Example 6-27 Contents of relational address table*

```
CUSTNAME    STRTNM         BLDGNB  PSTCD   TWNNM
----------- -------------- ------- ------- ---------
FINPETROL   Bailey Avenue  555     95141   San Jose
WINGPETROL  Bond Street    23      98282   New York
TAILPETROL  Southfork      1       99999   Dallas
```

There are many ways to perform an XML to relational join in SQL/XML. One approach is to use the XMLEXISTS function (which is indexable). A very simple join statement is illustrated in Example 6-28. The join is achieved by passing the relational column for the join (a.custname) to the XMLEXISTS function as a variable that is then used as an XML predicate.

*Example 6-28 XML to relational join using XMLEXISTS*

```
select c.msg_id, c.msg_cre_dt_tm, a.strtnm, a.bldgnb, a.pstcd, a.twnnm
from BK_TO_CSTMR_STMT c, ADDRESS a
   where xmlexists('declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $i/Document/BkToCstmrStmt/Stmt/Acct/Ownr[Nm=$acctname]'
      passing
         c.BK_TO_CSTMR_STMT as "i",
         a.custname as "acctname");

--yields

MSG_ID              MSG_CRE_DT_TM          STRTNM        BLDGNB PSTCD TWNNM
------------------- ---------------------- ------------- ------ ----- --------
AAAASESS-FP-STAT001 2010-01-15-17.00.00... Bailey Avenue 555    95141 San Jose
```

Another method of performing an XML to the relational join is using XMLTABLE. Example 6-29 shows normal relational join being coded between the result of an XMLTABLE function and a relational table. Example 6-29 does not have a predicate on the XMLTABLE function, so this particular query is not indexable.

*Example 6-29 XML to relational join using XMLTABLE*

```
SELECT X.ACCT_NM, X.SVCR_NM, a.strtnm, a.bldgnb, a.pstcd, a.twnnm
   FROM
      BK_TO_CSTMR_STMT as C,
      ADDRESS as a,
      XMLTable(XMLNAMESPACES(
         DEFAULT'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02'),
         '$d/Document/BkToCstmrStmt/Stmt/Acct'
            PASSING c.BK_TO_CSTMR_STMT as "d"
         COLUMNS "ACCT_NM"    VARCHAR(35)   PATH './Ownr/Nm',
         "SVCR_NM"    VARCHAR(35)    PATH './Svcr/FinInstnId/Nm' ) AS X
   where x.ACCT_NM = a.CUSTNAME ;

-- yields

ACCT_NM   SCVR_NM     STRTNM        BLDGNB PSTCD TWNNM
--------- ----------- ------------- ------ ----- --------
```

```
FINPETROL AAAA BANKEN Bailey Avenue 555    95141 San Jose
```

## 6.7.2 XML to XML join

The techniques that are required to perform XML to XML join differ slightly. To provide a test case, we convert the relational address table (from the XML to relational join examples in 6.7.1, "XML to relational join" on page 118) to an XML format. This conversion allows us to perform the same logical joins, except that both sources are XML.

The relational address table is converted to XML using the script in Example 6-30.

*Example 6-30 Script to convert the relational address table to XML*

```
create table xmladdress ( CUSTADDR XML ) ;

INSERT INTO XMLADDRESS (CUSTADDR)
   SELECT XMLSERIALIZE(XMLDOCUMENT(
      XMLELEMENT(NAME "MsgRcpt",
      XMLELEMENT(NAME "Nm", CUSTNAME),
      XMLELEMENT(NAME "PstlAdr",
         XMLELEMENT(NAME "StrtNm", STRTNM),
         XMLELEMENT(NAME "BldgNb", BLDGNB),
         XMLELEMENT(NAME "PstCd", PSTCD),
         XMLELEMENT(NAME "TwnNm", TWNNM) ))) AS CLOB)
from Address ;

select * from xmladdress ;

-- yields

CUSTADDR
--------------------------------------------------------------------
<MsgRcpt><Nm>FINPETROL</Nm><PstlAdr><StrtNm>Bailey
Avenue</StrtNm><BldgNb>555</BldgNb><PstCd>95141</PstCd><TwnNm>San
Jose</TwnNm></PstlAdr></MsgRcpt>

<MsgRcpt><Nm>WINGPETROL</Nm><PstlAdr><StrtNm>Bond
Street</StrtNm><BldgNb>23</BldgNb><PstCd>98282</PstCd><TwnNm>New
York</TwnNm></PstlAdr></MsgRcpt>

<MsgRcpt><Nm>TAILPETROL</Nm><PstlAdr><StrtNm>Southfork</StrtNm><BldgNb>1</BldgNb><
PstCd>99999</PstCd><TwnNm>Dallas</TwnNm></PstlAdr></MsgRcpt>
```

Having converted the address table to XML, we can now code XML to XML joins. Example 6-31 on page 121 shows the use of an XMLEXISTS function to join the two tables. In this example, we must pass both XML documents (`"i"` and `"j"`) into the XMLEXISTS function to perform the comparison.

The XML string function is used with the current location to signify that we want to perform a string comparison to join the current contents of one location in one document with the current contents of another location in the other document. The specification of the data type being used in the join comparison is important for the following reasons:

- Unlike relational columns, the XML fields do not have types defined.
- Index eligibility is dependent on the data type.

*Example 6-31 XML to XML join using XMLEXISTS*

```
select * from BK_TO_CSTMR_STMT c, XMLADDRESS a
   where xmlexists('declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $i/Document/BkToCstmrStmt/Stmt/Acct/Ownr[
         Nm/fn:string(.) = $j/MsgRcpt[ Nm/fn:string(.) ] ]'
      passing
         c.BK_TO_CSTMR_STMT as "i",
         a.CUSTADDR as "j");
```

Example 6-31 was created to illustrate the mechanics of an XML to XML join as clearly as possible. This query does not contain any filter predicate that is indexable. However, the join predicate is eligible for index access because it has been expressed with the string() function, which is indexable.

Similarly, if you have XML to XML join on a numeric field, use `xs:double` so that an XML DECFLOAT index can be used for the join predicate.

# 6.8 XML with change data capture tools

The primary application scenario for this book is the ISO 20022 standard for banking messages.

Another common source of XML messages is the range of replication and event publishing tools that are used to capture changes from existing database, and publish change data capture (CDC) messages. The published messages can sometimes be used to replicate changes to another database (such as a data warehouse). Increasingly, *CDC messages* are commonly being used in event-driven systems. Changes to source data, which meets certain criteria (such as bank transfers that exceed a threshold value), can be routed to a workflow system (such as WebSphere Message Broker) where the CDC event can be examined by using workflows that implement business processes, and initiate automated actions.

This section considers ways in which CDC messages can be handled by DB2 pureXML.

## 6.8.1 Change data capture tools background

Change data capture tools tend to follow an architecture along the following lines:

1. A *capture* process is used to read the database log of a source database, looking for changes that have been requested by a subscription definition.
2. When qualifying changes are found in the log, they are packaged (typically into unit-of-work boundaries) and transmitted over a network infrastructure to the target systems that have subscribed to them.
3. The target systems receive the changes and do something with them (such as update a database, invoke an application process, publish an CDC message in XML or other format, and so on).

IBM replication and event publishing tools that publish XML CDC messages include the tools listed in Table 6-3 on page 122. In the table, LUW stands for Linux, UNIX, and Windows.

*Table 6-3 IBM change data capture tools that publish XML messages.*

| Tool | Main data sources supported for CDC | Comments |
|---|---|---|
| InfoSphere Data Event Publisher | ► DB2 for z/OS<br>► DB2 for LUW<br>► Oracle | ► Offers asynchronous log reader services<br>► Writes CDC messages as XML (and other formats)<br>► Publishes messages directly to WebSphere MQ<br>► Is available for z/OS and LUW versions |
| InfoSphere Classic Data Event Publisher | ► IMS<br>► VSAM<br>► IDMS<br>► Adabas | ► Offers asynchronous log reader services<br>► Writes CDC messages as XML (and other formats)<br>► Publishes messages directly to WebSphere MQ or zFS files<br>► Is available for z/OS only. |
| InfoSphere Change Data Capture | ► DB2 for z/OS<br>► DB2 for LUW<br>► DB2 for iSeries®<br>► Oracle<br>► Sybase<br>► SQL Server | ► Offers asynchronous log reader services<br>► Writes CDC messages as XML (and other formats)<br>► Source server writes CDC data over tcpip to a target server.<br>► Target server writes messages to a range of targets, including WebSphere MQ and files<br>► Is available for z/OS and LUW versions |

The purpose of this section is to focus on the XML messages that are published from these tools, and examine how they can be used with DB2 pureXML.

InfoSphere Data Event Publisher and InfoSphere Classic Data Event Publisher both share a common schema and generate messages similar to the one in Example 6-32.

*Example 6-32 XML CDC message format for DB2 and Classic Data Event Publishers*

```
<?xml version="1.0" ?>
   <msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd"
      version="1.0.0" dbName="$IMS    "
      seqNum="IMS000003B4009FF00AIMSY         _8"> 1
      <rowOp  authID="USER0000"
         planName="USER0000"
         cmitLSN="IMS000003B4009FF00AIMSY         "
         cmitTime="2010-11-12:14:22:02.374839"> 2
      <insertRow subName="CLASSIC" srcOwner="IMSP" srcName="CLASSIC1"> 3
         <col name="STRTNM" isKey="0"> 4
            <char>Bailey Avenue</char>
         </col>
         <col name="BLDGNB" isKey="0">
            <char>555</char>
         </col>
         <col name="PSTCD" isKey="0">
            <char>95149</char>
         </col>
         <col name="TWNNM" isKey="0">
            <char>San Jose</char>
         </col>
      </insertRow>
      </rowOp>
   </msg>
```

The annotated lines in the example are as follows:

**1.** The XML schema is mqcap.xsd is available to be defined in the pureXML XSR. the document root also contains information about the database source.

**2.** CDC messages can be generated based either as row operations (such as this one) or as transactions. A transaction message inserts `<trans>` tags around one or more `<rowOp>` tags. log sequence numbers and timestamps are included as attributes, and reflect the information that is available for a particular data source.

**3.** Individual row operations may be `<InsertRow>`, `<UpdateRow>`, or `<DeleteRow>`.

**4.** The before and after images of the column values are published. The administration tools has options to ignore before images, and make other customization to the published data.

InfoSphere CDC offers the choice of a range of XML schemas for publishing change data messages. Example 6-33 shows an SQL update of a simple InfoSphere CDC XML message.

*Example 6-33 XML CDC message format for InfoSphere CDC*

```
<?xml version="1.0" encoding="UTF-8"?>
   <CUST_CHANGE>
      <TimeStamp AfterImage="2010-10-22T14.22.38.000"
         BeforeImage="2009-05-16T10.09.22.000"/>
      <SourceLogSeqNumber AfterImage="20101022142238000000"
         BeforeImage="20090516100922000000"/>
      <STRTNM AfterImage=" Nice Street " BeforeImage=" Bailey Avenue "/>
      <BLDGNB AfterImage=" 100 " BeforeImage=" 555 "/>
      <PSTCD AfterImage=" 98282 " BeforeImage=" 95141 "/>
      <TWNNM AfterImage=" San Diego " BeforeImage=" San Jose "/>
   </CUST_CHANGE>
```

Based on the XML examples presented in the previous sections in this chapter, these CDC XML messages look fairly straightforward to handle.

## 6.8.2 Using DB2 pureXML to receive CDC messages

Change data capture event messages are useful for many reasons, including the following reasons:

- Updating a data warehouse with a stream of change data messages, so that it can be kept up-to-date with very low latency compared to other techniques such as extracts and loads.
- Notifying workflow or event driven systems about important changes to operational data, that require rapid or automated business processes to be executed

Another potential use for CDC messages is to maintain a historical record of changes to operational data. We expand this idea in 6.8.3, "XML history objects" on page 129.

For now, we examine how to consume a stream of XML CDC messages with DB2 pureXML. We could use a mixture of all the programming constructs that we have illustrated in this chapter so far to receive CDC XML messages into DB2 pureXML storage, as illustrated in Figure 6-8 on page 124.

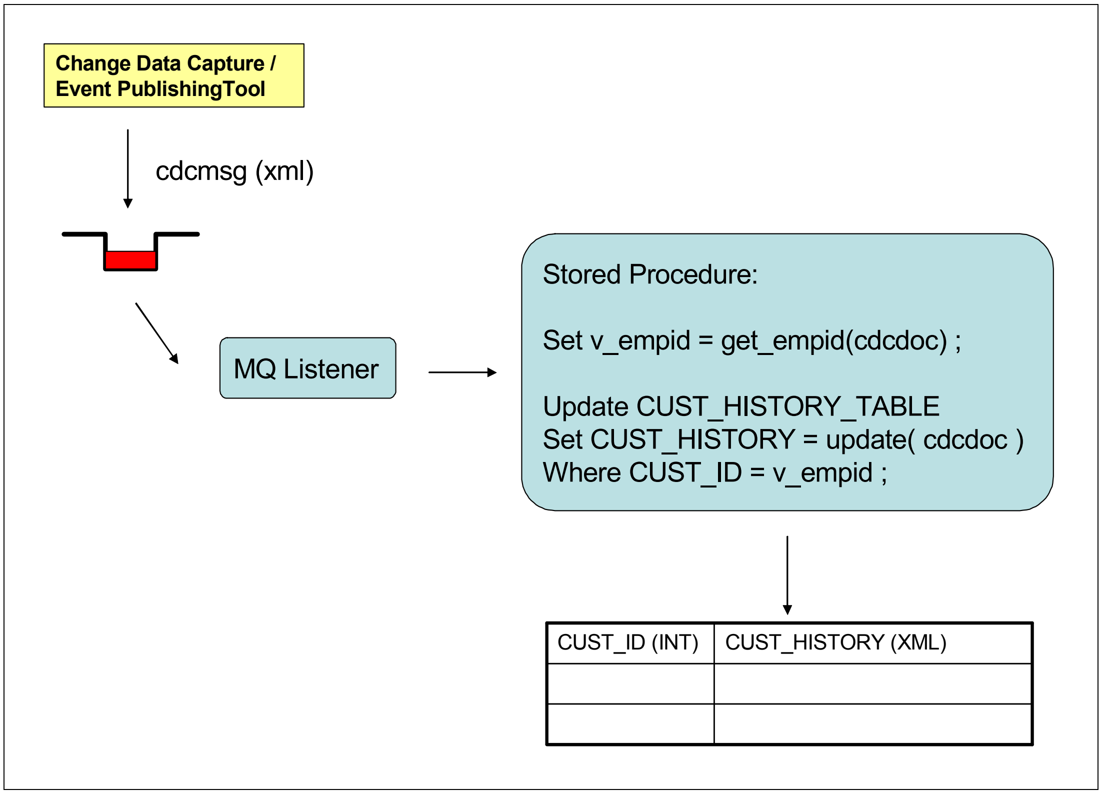
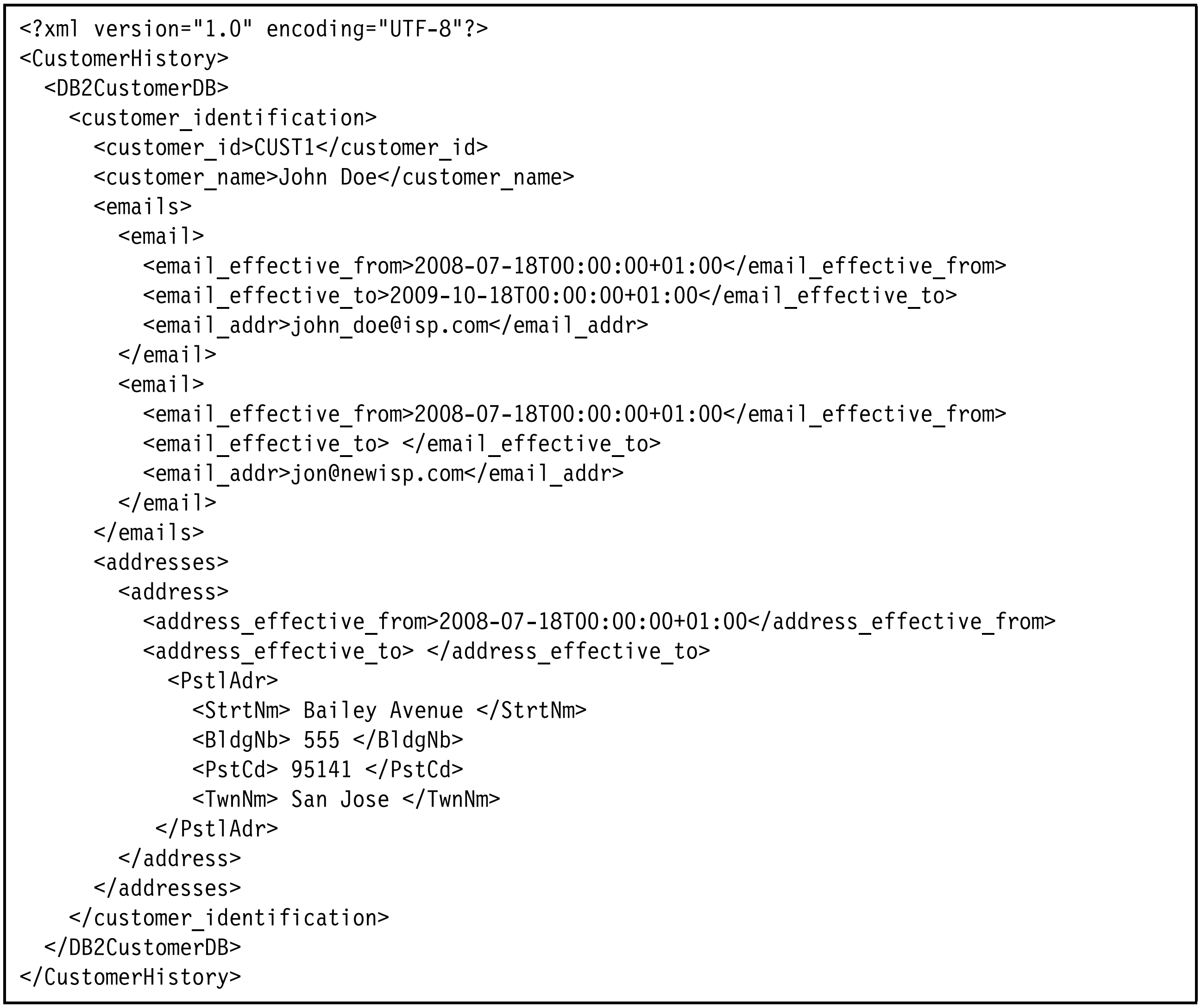

Change Data Capture / Event PublishingTool
cdcmsg (xml)
MQ Listener
Stored Procedure:
Set v_empid = get_empid(cdcdoc) ;
Update CUST_HISTORY_TABLE
Set CUST_HISTORY = update( cdcdoc )
Where CUST_ID = v_empid ;
CUST_ID (INT)
CUST_HISTORY (XML)

*Figure 6-8 Scenario to receive XML CDC messages into DB2 pureXML through MQ*

Previously in this chapter, We describe the mechanics of receiving XML messages from WebSphere MQ. The requirements are more complex if we want to update a historical audit record.

Figure 6-9 on page 125 shows the contents of the XML historical record before the CDC record is applied. Fictional customer John Doe, with customer number CUST1 has a record of two nominated email addresses (one expired, one active) and only one postal address (which is active). If we receive a CDC message containing a change of address, we must set an ending date in the node for the current postal address, and add a new node for the new address.



```
<?xml version="1.0" encoding="UTF-8"?>
<CustomerHistory>
  <DB2CustomerDB>
    <customer_identification>
      <customer_id>CUST1</customer_id>
      <customer_name>John Doe</customer_name>
      <emails>
        <email>
          <email_effective_from>2008-07-18T00:00:00+01:00</email_effective_from>
          <email_effective_to>2009-10-18T00:00:00+01:00</email_effective_to>
          <email_addr>john_doe@isp.com</email_addr>
        </email>
        <email>
          <email_effective_from>2008-07-18T00:00:00+01:00</email_effective_from>
          <email_effective_to> </email_effective_to>
          <email_addr>jon@newisp.com</email_addr>
        </email>
      </emails>
      <addresses>
        <address>
          <address_effective_from>2008-07-18T00:00:00+01:00</address_effective_from>
          <address_effective_to> </address_effective_to>
            <PstlAdr>
              <StrtNm> Bailey Avenue </StrtNm>
              <BldgNb> 555 </BldgNb>
              <PstCd> 95141 </PstCd>
              <TwnNm> San Jose </TwnNm>
           </PstlAdr>
        </address>
      </addresses>
    </customer_identification>
  </DB2CustomerDB>
</CustomerHistory>
```

*Figure 6-9 Initial CUST_HISTORY table contents for CUST1*

If we receive the CDC message in Example 6-33 on page 123, we must write a procedure that performs the following steps:

1. Examine the contents of the XML CDC record to find the customer identifier key.
2. Use the customer identifier key to retrieve the XML document that stores the historical record of customer address changes.
3. Update the XML node in the document that stored the current address details, to set an end date for that address.
4. Insert a new node in the XML document to reflect the new address details.

The stored procedure in Example 6-34 on page 126 receives the XML CDC message from MQ (in the format of the InfoSphere CDC example), and updates the historical record of changes in DB2. Although this stored procedure picks up the CDC message from a test table, MQ examples earlier in this chapter show how to adapt the procedure to work with WebSphere MQ.

*Example 6-34 Stored procedure to receive and apply CDC message*

```
CREATE PROCEDURE XMLR3.RECEIVE_CDC(IN CDCDOC XML) 1
   LANGUAGE SQL
   MODIFIES SQL DATA
   DISABLE DEBUG MODE

BEGIN

DECLARE CDC_CUSTID CHAR(5) ;
DECLARE CDC_STRTNM VARCHAR(70) ;
DECLARE CDC_BLDGNB VARCHAR(16) ;
DECLARE CDC_PSTCD VARCHAR(16)  ;
DECLARE CDC_TWNNM VARCHAR(35)  ;
DECLARE V_ADDR_FROM TIMESTAMP WITH TIMEZONE ;
DECLARE V_STRTNM VARCHAR(70) ;
DECLARE V_BLDGNB VARCHAR(16) ;
DECLARE V_PSTCD VARCHAR(16) ;
DECLARE V_TWNNM VARCHAR(35) ;
DECLARE PREV_MAX_ADDR XML ;
DECLARE NEXT_MAX_ADDR XML ;
DECLARE UDF_RETURN1 INT ;
DECLARE UDF_RETURN2 INT ;

SELECT X.CUSTID, X.STRTNM, X.BLDGNB, X.PSTCD, X.TWNNM
   INTO CDC_CUSTID, CDC_STRTNM, CDC_BLDGNB, CDC_PSTCD, CDC_TWNNM
   FROM XMLTable('$d/CUST_CHANGE' PASSING CDCDOC as "d"
      COLUMNS
         "CUSTID"   CHAR(5) PATH 'CUSTID/@AfterImage',
         "STRTNM"   VARCHAR(70) PATH 'STRTNM/@AfterImage',
         "BLDGNB"   VARCHAR(16) PATH 'BLDGNB/@AfterImage',
         "PSTCD"    VARCHAR(16) PATH 'PSTCD/@AfterImage',
         "TWNNM"    VARCHAR(35) PATH 'TWNNM/@AfterImage'
    ) AS X ; 2

SELECT X.ADDR_FROM, X.STRTNM, X.BLDGNB, X.PSTCD, X.TWNNM
   INTO V_ADDR_FROM, V_STRTNM, V_BLDGNB, V_PSTCD, V_TWNNM
   FROM CUST_HISTORY C,
   XMLTable(
   '$cu/CustomerHistory/DB2CustomerDB/customer_identification/addresses/address'
   PASSING C.CUST_HISTORY_OBJECT as "cu"
      COLUMNS
         "ADDR_FROM" TIMESTAMP WITH TIMEZONE  PATH 'address_effective_from',
         "STRTNM"    VARCHAR(50)              PATH 'PstlAdr/StrtNm',
         "BLDGNB"    VARCHAR(50)              PATH 'PstlAdr/BldgNb',
         "PSTCD"     VARCHAR(50)              PATH 'PstlAdr/PstCd',
         "TWNNM"     VARCHAR(50)              PATH 'PstlAdr/TwnNm'
   ) AS X
      WHERE CUSTID = CDC_CUSTID
      order by X.ADDR_FROM desc
         fetch first row only ; 3

SET PREV_MAX_ADDR =  XMLELEMENT(NAME "address",
      XMLELEMENT(NAME "address_effective_from", V_ADDR_FROM),
      XMLELEMENT(NAME "address_effective_to", current timestamp with timezone),
      XMLELEMENT(NAME "PstlAdr",
          XMLELEMENT(NAME "StrtNm", V_STRTNM),
          XMLELEMENT(NAME "BldgNb", V_BLDGNB),
          XMLELEMENT(NAME "PstCd", V_PSTCD),
          XMLELEMENT(NAME "TwnNm", V_TWNNM)))  ; 4

SET NEXT_MAX_ADDR =  XMLELEMENT(NAME "address",
      XMLELEMENT(NAME "address_effective_from", current timestamp with timezone),
      XMLELEMENT(NAME "address_effective_to", ''),
      XMLELEMENT(NAME "PstlAdr",
          XMLELEMENT(NAME "StrtNm", CDC_STRTNM),
          XMLELEMENT(NAME "BldgNb", CDC_BLDGNB),
          XMLELEMENT(NAME "PstCd", CDC_PSTCD),
          XMLELEMENT(NAME "TwnNm", CDC_TWNNM))) ; 5

UPDATE CUST_HISTORY X
   SET X.CUST_HISTORY_OBJECT = XMLMODIFY(
      'replace node
        /CustomerHistory/DB2CustomerDB/customer_identification/addresses/address[
        address_effective_from=$t]
      with $V',
      PREV_MAX_ADDR AS "V",
      V_ADDR_FROM as "t" )
   where CUSTID = 'CUST1' ; 6

UPDATE CUST_HISTORY X
   SET X.CUST_HISTORY_OBJECT = XMLMODIFY(
      'insert node $new as last into
         /CustomerHistory/DB2CustomerDB/customer_identification/addresses',
      NEXT_MAX_ADDR as "new" )
   where CUSTID = 'CUST1' ; 7

END  !
```

The logic of the stored procedure is as follows, with the annotated points from Example 6-34 on page 126.

1. The stored procedure accepts the CDC message as an XML document in INPUT parameter CDCDOC.
2. The data elements of the incoming CDC message are stripped with a single XMLTABLE function.
3. Using the customer ID (‘CUST1’) from the CDC document, the current address node is retrieved from the CUSTOMER_HISTORY table.
4. A replacement XML node is derived for the current address node, using XML publishing functions.
5. A new XML node is derived from the incoming CDCDOC.
6. The existing XML node that is storing the current address is replaced by using an XMLMODIFY function.
7. The new XML node is inserted by using a further XMLMODIFY function.

PREV_MAX_ADDR and NEXT_MAX_ADDR are kept as XML types because they will be used in XMLMODIFY later, therefore avoiding an XMLPARSE() method.

The resultant contents of the addresses nodes in the customer history table is shown in Figure 6-10.

```
<addresses>
  <address>
    <address_effective_from>2008-07-18T00:00:00.000000+01:00</address_effective_from>
    <address_effective_to>2010-11-08T17:31:03.602776-05:00</address_effective_to>
    <PstlAdr>
      <StrtNm> Bailey Avenue </StrtNm>
      <BldgNb> 555 </BldgNb>
      <PstCd> 95141 </PstCd>
      <TwnNm> San Jose </TwnNm>
   </PstlAdr>
  </address>
  <address>
    <address_effective_from>2010-11-08T17:31:03.602935-05:00</address_effective_from>
    <address_effective_to/>
    <PstlAdr>
      <StrtNm> Nice Street </StrtNm>
      <BldgNb> 100 </BldgNb>
      <PstCd> 98282 </PstCd>
      <TwnNm> San Diego </TwnNm>
    </PstlAdr>
  </address>
</addresses>
```

*Figure 6-10 Updated CUST_HISTORY table contents for ‘CUST1’*

If you use the InfoSphere Data Event Publisher or Classic Data Event Publisher products, the XML schema differs, although the technique is the same. All you need to do is replace the XMLTABLE operation on the incoming XML CDC document with a separate XMLTABLE operation, shown in Example 6-35, to work with the incoming schema.

*Example 6-35 XMLTABLE function for Event Publisher XML schemas*

```
SELECT X.CUSTID, X.STRTNM, X.BLDGNB, X.PSTCD, X.TWNNM
   INTO CDC_CUSTID, CDC_STRTNM, CDC_BLDGNB, CDC_PSTCD, CDC_TWNNM
      FROM XMLTable('$d/msg/rowOp/updateRow' PASSING CDCDOC as "d"
      COLUMNS
         "CUSTID"   CHAR(5) PATH     'col[@name=”CUSTID”]/char/afterVal',
         "STRTNM"   VARCHAR(70) PATH 'col[@name=”STRTNM”]/varchar/afterVal',
         "BLDGNB"   VARCHAR(16) PATH 'col[@name=”BLDGNB”]/varchar/afterVal',
         "PSTCD"    VARCHAR(16) PATH 'col[@name=”PSTCD”]/varchar/afterVal',
         "TWNNM"    VARCHAR(35) PATH 'col[@name=”TWNNM”]/varchar/afterVal'
      ) AS X  ;
```

**Note:** Apply the program temporary fix (PTF) for APAR PM28385 (currently open) for the most recent maintenance that is related to XMLTABLE function.

## 6.8.3 XML history objects

This section explores the use of pureXML to store historical data, and support temporal queries against that data. We consider an option to create and maintain pureXML data objects (representing a customer, for example) that contain a growing historical record of all changes that have been applied to those objects in the source systems.

### Bitemporal data

Before examining XML support for temporal data, we first investigate another DB2 10 capability that supports the storage of historical data and the execution of temporal queries against that data: *bitemporal data*.

The concept of bitemporal data is that four more columns (timestamp data type) are added to tabular data, reflecting the starting and ending points of *system time* and *business time*:

- System time is an auditable history of what the data looked like in the system at any point in history.
- Business time is an auditable history of the data, which reflects business corrections.

Bitemporal data support is a productivity feature of DB2 that can be used to maintain an auditable history of changes over time. Whenever a change is made to a table with bitemporal data support, the system time and the business time are both automatically tracked to maintain that auditable history.

Bitemporal data is not explained in any more detail in this book, because it is covered in other DB2 10 publications. For new DB2 systems, particularly for the relational data model, bitemporal data probably is the most productive way of supporting temporal data, because the data structures and programming interface have all been designed into DB2. Also, the use of bitemporal data in conjunction with XML is fully supported.

### Temporal data with XML

XML also provides support for temporal data.

The value of XML for historical data is broadly acknowledged as powerful and efficient. The paper *XML-Based Support for Database Histories and Document Versions* by Fusheng Wang, 2004, describes an XML model for storing historical data, and supporting temporal queries. It is available at the following address:

`http://wis.cs.ucla.edu/~wangfsh/publications/thesis.pdf`

### Building XML History Objects from change data capture data streams

Most legacy data sources have been defined without system-provided temporal data support because bitemporal data is new in DB210 for z/OS.

Many transactional database systems are focussed almost exclusively on the current state of the data that they store, and might not have much support for historical data. This scenario is common and often leads to a decision to deploy a data warehouse (which is derived from the data in the transactional system) for the purpose of storing historical data and performing trend analysis to see how data has changed over time.

CDC and replication products are often used as a vehicle to feed changes to data warehouses. The CDC data streams contain details of what the data changes were, and precisely when they happened, usually as a commit timestamp from the log record on the source system. This temporal reference data can be used by the extract, transform, and load (ETL) processes that maintain the data warehouse, to build a record of historical changes in the target data warehouse.

Consider DB2 pureXML as a data type in the data warehouse for recording a history of changes to source data objects. For data warehouses that seek to deliver OLAP-style structures, pureXML is not a good choice. However, pureXML is a good choice for other scenarios where the state of data at various past historical points is of interest. The following characteristics tend to make pureXML attractive for historical data:

- Large and sparse data structures could require extensive and costly relational database structures, but can be implemented with simplicity within an XML schema
- sub-document update allows changes replicated from source systems to be merged into a singe XML document, combining all historical changes into a single XML document (as shown in Example 6-34 on page 126)
- SQL/XML provides powerful temporal query capabilities (provided the XML schema is designed to be friendly for temporal queries)
- XML and relational storage can co-exist in DB2's hybrid database environment, supporting queries with a mixture of SQL and XML expressions, and a mixture of relational and XML storage.

DB2 pureXML could act as a repository for historical data, with temporal query support as in Figure 6-11.

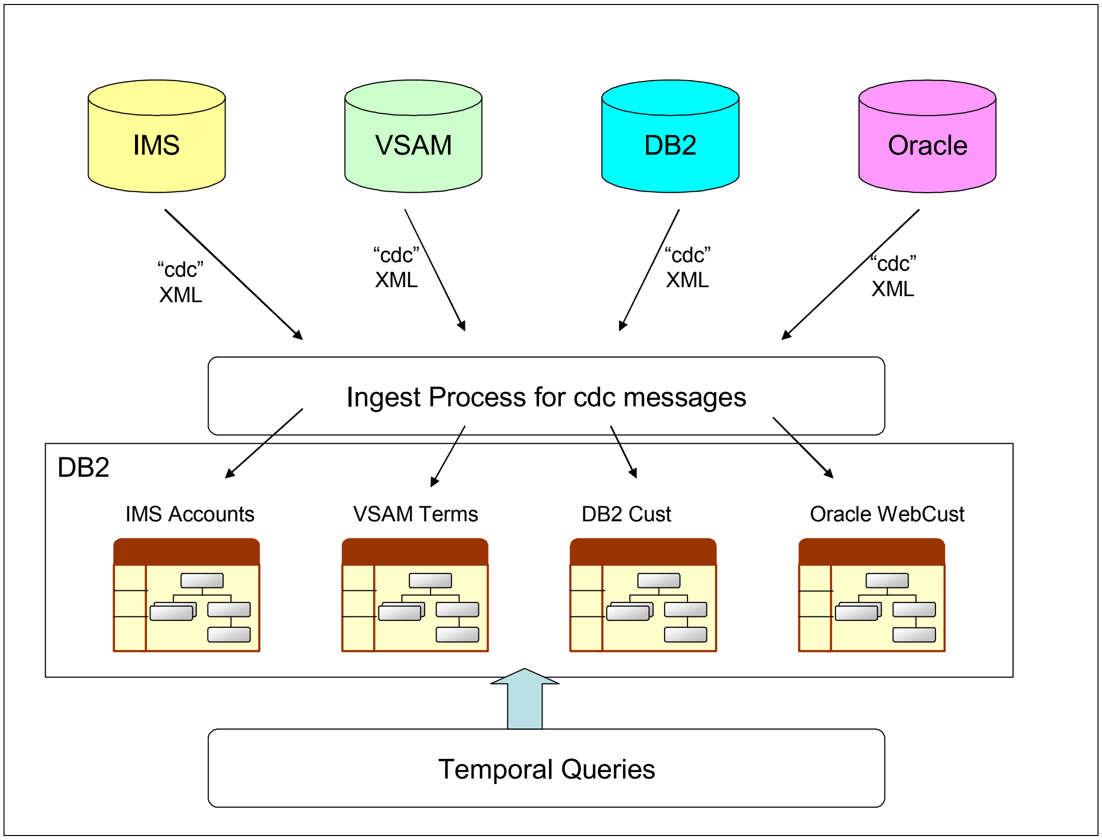


*Figure 6-11 DB2 pureXML as historical repository*

7

# Using XML with Java

In Java applications, you can store XML data in DB2 databases or retrieve XML data from DB2 databases by using JDBC or SQLJ interface. Java also provides powerful APIs for XML processing, such as DOM, SAX, and StAX.

In this chapter, we show how JDBC applications handle XML data in DB2 for z/OS, based on a scenario that uses the ISO 20022 BankToCustomerStatement message.

This chapter contains the following topics:

- XML in Java
- The BankStmt application in Java

The description of the application scenario is in Chapter 3, “Application scenario” on page 47.

All Java examples are available for download as additional material. See Appendix B, “Additional material” on page 277.

© Copyright IBM Corp. 2011. All rights reserved.

## 7.1 XML in Java

In DB2 tables, the built-in XML data type is used to store XML data in a column as a structured set of nodes in a tree format. You may write applications to store XML data in DB2 tables and retrieve XML data from tables.

The Java programming language and its database interface JDBC are popular choices for XML application development. DB2 for z/OS provides a universal driver that supports both the JDBC and the SQLJ interface of the Java language. This driver is the IBM Data Server Driver for JDBC and SQLJ, also known as Java Common Client (JCC). It supports JDBC type 2 and type 4 driver and can connect to the DB2 family of products and Informix® Dynamic Server database systems.

Two versions of the IBM Data Server Driver for JDBC and SQLJ are available:

- Version 3.x, JDBC 3.0-compliant
- Version 4.x, JDBC 4.0-compliant

Table 7-1 summarizes the Java interfaces for XML data type support.

*Table 7-1 XML data type support in JCC3 and JCC4*

| JCC driver | JDBC support | Java interface for XML data | Minimum Java level required | JAR file |
|---|---|---|---|---|
| JCC 3.x | JDBC 3.0 | com.ibm.db2.jcc.DB2Xml | 1.4 | db2jcc.jar<br>sqlj.jar |
| JCC 4.x | JDBC 4.0 | java.sql.SQLXML | 6.0 | db2jcc4.jar<br>sqlj4.jar |

You control the level of JDBC support by specifying the appropriate set of files in the CLASSPATH. For example, if you want to use the JCC 4.x driver, include `lib_dir/db2jcc4.jar` and `lib_dir/sqlj4.jar` files in your CLASSPATH.

In JDBC applications, you can perform the following tasks:

- Register and remove XML schemas using Java methods.
- Create XML documents from application data using XML APIs.
- Store an entire XML document in an XML column using setXXX methods.
- Retrieve an entire XML document from an XML column using getXXX methods.
- Retrieve a sequence from a document in an XML column by using the SQL XMLQUERY function to retrieve the sequence into a serialized sequence in the database, and then using getXXX methods to retrieve the data into an application variable.
- Retrieve a sequence from a document in an XML column as a user-defined table by using the SQL XMLTABLE function to define the result table and retrieve it. Then, use getXXX methods to retrieve the data from the result table into application variables.
- Invoke routines with XML parameters in Java applications.

JDBC3.0 provides basic methods to retrieve an XML document as Bytes, String, or Stream, or set the XML document from Bytes, String, or Stream. IBM Data Server Driver for JDBC and SQLJ 3.x also supports DB2Xml interfaces for XML processing.

JDBC4.0 introduces the SQLXML Java data type, which you use to map an XML data type table column to a Java data type. Use IBM Data Server Driver for JDBC and SQLJ 4.x, the latest level. This driver provides the SQLXML interface and also support other DB2 features such as binary XML format. You have more flexibility in coding your application and better performance.

### 7.1.1 XML support in JDBC 3.0

When developing applications with JDBC3.0, you can use the Java standard interface, ResultSet, to retrieve XML data from DB2 tables. This interface provides getter methods for retrieving XML column values from the current row. Table 7-2 summarizes the methods in the ResultSet interface.

*Table 7-2 JDBC 3.0 Getter methods of ResultSet*

| Getter methods of ResultSet | Description |
|---|---|
| getBytes | Retrieves the value of the designated column in the current row of this ResultSet object as UTF-8 encoded bytes. |
| getString | Retrieves the value of the designated column in the current row of this ResultSet object as a string in the Java programming language. |
| getAsciiStream | Retrieves the value of the designated column in the current row of this ResultSet object as a stream of ASCII characters. |
| getBinaryStream | Retrieves the value of the designated column in the current row of this ResultSet object as a UTF-8 encoded binary stream. |
| getCharacterStream | Retrieves the value of the designated column in the current row of this ResultSet object as a `java.io.Reader` object. |
| getObject | Retrieves the value of the designated column in the current row of this ResultSet object as a `com.ibm.db2.jcc.DB2Xml` object. |

**Note:** The methods in Table 7-2 do not add an encoding declaration to the retrieved XML data.

The getObject method retrieves XML data into an object of type DB2Xml, which provides more getter methods. The `com.ibm.db2.jcc.DB2Xml` object supports the methods that are listed in Table 7-3.

*Table 7-3 DB2Xml getter methods*

| DB2Xml getter methods | Description |
|---|---|
| getDB2Bytes() | Retrieves the value of the designated column in the current row of this ResultSet object as UTF-8 encoded bytes. |
| getDB2XmlBytes() | Retrieves the value of the designated column in the current row of this ResultSet object as a byte array in the Java programming language. The method converts the bytes to the target encoding and adds XML declaration with encoding tag. |
| getDB2String() | Retrieves the value of the designated column in the current row of this ResultSet object as a string in the Java programming language. |

| DB2Xml getter methods | Description |
|---|---|
| getDB2XmlString() | Retrieves the value of the designated column in the current row of this ResultSet object as a string in the Java programming language. The method adds an XML declaration with the `"ISO-10646-UCS-2"` encoding tag. |
| getDB2AsciiStream() | Retrieves the value of the designated column in the current row of this ResultSet object as a stream of ASCII characters. |
| getDB2XmlAsciiStream() | Retrieves the value of the designated column in the current row of this ResultSet object as a stream of ASCII characters. The method will add XML declaration with encoding tag |
| getDB2BinaryStream() | Retrieves the value of the designated column in the current row of this ResultSet object as a UTF-8 encoded binary stream. |
| getDB2XmlBinaryStream() | Retrieves the value of the designated column in the current row of this ResultSet object as a binary stream. The method converts the bytes to the target encoding and adds the XML declaration with encoding tag. |
| getDB2CharacterStream() | Retrieves the value of the designated column in the current row of this ResultSet object as a `java.io.Reader` object |
| getDB2XmlCharacterStream() | Retrieves the value of the designated column in the current row of this ResultSet object as a `java.io.Reader` object. The method adds an XML declaration with `"ISO-10646-UCS-2"` encoding tag. |

The getDB2XmlXXX methods generate XML declarations with an encoding attribute for the retrieved XML data. For example, the getDB2String() and getDB2XmlString() methods return the XML data in the same encoding, USC-2, however the latter adds the appropriate encoding declaration to the XML document.

In a JDBC application, you can update or insert data into XML columns of a table at a DB2 data server using one of the setter methods of the PreparedStatement interface.

The following setXXX methods are supported against XML columns in JDBC 3.0:

- setAsciiStream()
- setBinaryStream()
- setBlob()
- setBytes()
- setCharacterStream()
- setClob()
- setString()
- setObject()

  This method supports DB2Xml, String, byte[], InputStream, Reader, CLOB, and BLOB as parameters.

### 7.1.2 XML support in JDBC 4.0

In JDBC 4.0, the main feature for XML support is the SQLXML object, which is the mapping in the Java programming language for the SQL XML data type. The SQLXML interface provides methods for accessing the XML value as a String, a Reader or Writer, or as a Stream. The XML value may also be accessed through a Source or set as a Result, which are used with XML Parser APIs such as DOM, SAX, and StAX.

The interfaces ResultSet, CallableStatement, and PreparedStatement are enhanced with new getter or setter methods, such as `ResultSet.getSQLXML`, to allow a programmer to retrieve the value of the designated column of this ResultSet as a `java.sql` SQLXML object in the Java programming language.

The SQLXML interface provides methods, listed in Table 7-4, for retrieving XML data from an SQLXML object.

*Table 7-4 Methods to retrieve XML data from SQLXML object*

| SQLXML getter methods | Description |
|---|---|
| getBinaryStream() | Retrieves the XML value designated by this SQLXML instance as a stream. |
| getCharacterStream() | Retrieves the XML value designated by this SQLXML instance as a `java.io.Reader` object. |
| getString() | Returns a string representation of the XML value designated by this SQLXML instance. |
| getSource(Class<T> sourceClass) | Returns a source for reading the XML value designated by this SQLXML instance. |

The SQLXML interface provides methods, listed in Table 7-5, for setting XML data to a JDBC object.

*Table 7-5 Method to set XML value to SQLXML object*

| SQLXML setter methods | Description |
|---|---|
| setBinaryStream() | Retrieves a stream that can be used to write the XML value that this SQLXML instance represents. |
| setCharacterStream() | Retrieves a stream to be used to write the XML value that this SQLXML instance represents. |
| setString() | Sets the XML value designated by this SQLXML instance to the given String representation. |
| setResult(Class<T> resultClass) | Returns a Result for setting the XML value designated by this SQLXML instance.<br>void setString(String value) |

For the implementation of the BankStmt application in Java, we mostly use the JDBC 4.0 standard SQLXML object in our programming.

### 7.1.3 Constructing XML document in Java

You can construct XML document by using DB2 SQL/XML publishing functions. If your application holds information in application variables and wants to construct this data into an XML document, you can also do this task by writing Java code with separate XML APIs. In our example, we construct a GroupHeader (subdocument of the BankToCustomerStatement message). We use DOM APIs in this example because it is easy to understand. The GroupHeader we want to create is shown in Example 7-1.

*Example 7-1 Creating a GroupHeader of the BankToCustomerStatement message*

```
<GrpHdr>
   <MsgId>AAAASESS-FP-ACCR001</MsgId>
   <CreDtTm>2010-10-18T12:30:00+01:00</CreDtTm>
   <MsgPgntn>
      <PgNb>1</PgNb>
      <LastPgInd>true</LastPgInd>
   </MsgPgntn>
</GrpHdr>
```

The code for constructing XML as a DOM tree is shown in Example 7-2.

*Example 7-2 Constructing XML as a DOM tree*

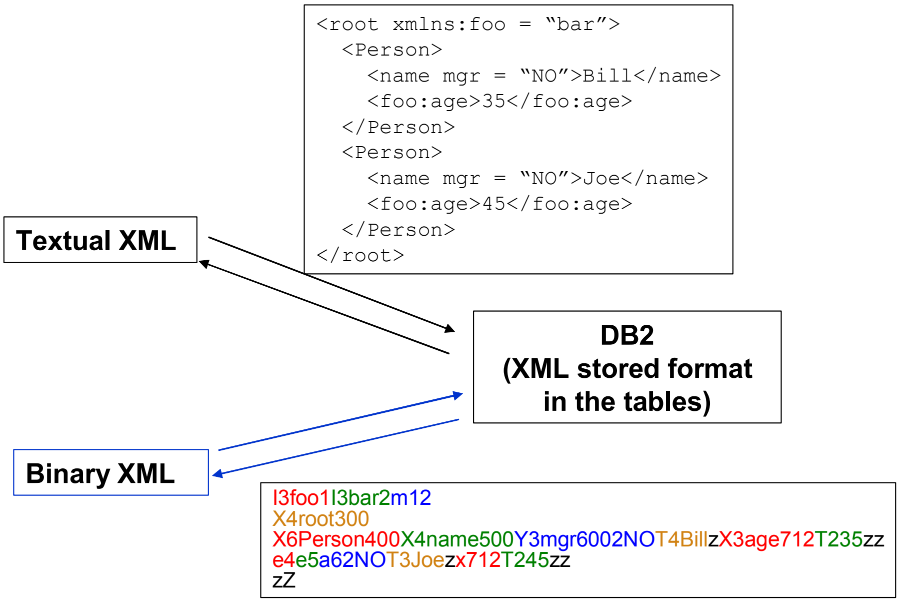
```
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Text;
...
String MsgIdStr="AAAASESS-FP-ACCR001";
String CreDtTmStr="2010-10-18T12:30:00+01:00";
String PgNbStr="1";
String LastPgIndStr="true";

DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();

try{
   DocumentBuilder builder = factory.newDocumentBuilder();
   Document document = builder.newDocument(); 1

   //************************************************************************
   // Create element nodes
   //************************************************************************
   Element GrpHdr = document.createElement("GrpHdr"); 2
   Element MsgId = document.createElement("MsgId");
   Element CreDtTm = document.createElement("CreDtTm");
   Element MsgPgntn = document.createElement("MsgPgntn");
   Element PgNb = document.createElement("PgNb");
   Element LastPgInd = document.createElement("LastPgInd");

   //************************************************************************
   // Create text nodes with designated values
   //************************************************************************
   Text MsgIdValue = document.createTextNode(MsgIdStr); 3
   Text CreDtTmValue = document.createTextNode(CreDtTmStr);
   Text PgNbValue = document.createTextNode(PgNbStr);
   Text LastPgIndValue = document.createTextNode(LastPgIndStr);

   //**************************************************************************
   // Append text node under element nodes
   //**************************************************************************
   MsgId.appendChild(MsgIdValue); 4
   CreDtTm.appendChild(CreDtTmValue);
   PgNb.appendChild(PgNbValue);
   LastPgInd.appendChild(LastPgIndValue);

   //**************************************************************************
   // Construct the DOM tree
   //**************************************************************************
   MsgPgntn.appendChild(PgNb); 5
   MsgPgntn.appendChild(LastPgInd);
   GrpHdr.appendChild(MsgId);
   GrpHdr.appendChild(CreDtTm);
   GrpHdr.appendChild(MsgPgntn);

   document.appendChild(GrpHdr);

}catch (Exception e) {
   System.out.println("Some exception occur: " +
      e.getMessage());
}

Note:
1.Obtain a new instance of a DOM Document object to build a DOM tree
2.Document.createElement() method to create element nodes with specified tagName
3.Document.createTextNode() method to create text nodes with designated values
4.Element.appendChild() method to append text node under element nodes
5.Append element nodes and construct the DOM tree
```

### 7.1.4 Binary XML format in Java applications

DB2 10 for z/OS introduces a binary format for XML data to be used with Java applications, which is called Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format (XDBX)[1]. This format is an external representation of an XML value that is used only for exchange with a DB2 client application or the UNLOAD or LOAD utilities. The binary representation is smaller in size, and it saves the parsing cost.

The IBM Data Server Driver for JDBC and SQLJ can send XML data to the data server or retrieve XML data from the data server as textual XML data or binary XML data, as shown in Figure 7-1 on page 138.

[1] http://www.ibm.com/support/docview.wss?uid=swg27019354&aid=1



```
<root xmlns:foo = "bar">
  <Person>
    <name mgr = "NO">Bill</name>
    <foo:age>35</foo:age>
  </Person>
  <Person>
    <name mgr = "NO">Joe</name>
    <foo:age>45</foo:age>
  </Person>
</root>
```

```
I3foo1I3bar2m12
X4root300
X6Person400X4name500Y3mgr6002NOT4BillzX3age712T235zz
e4e5a62NOT3Joezx712T245zz
zZ
```

- Binary XML is smaller in size
- Savings in DB2 CPU time during insert and select
- Savings in time end to end for insert

*Figure 7-1 Exchange data as textual or binary XML format*

You may set the JCC data source property, xmlFormat, to control whether the data format is textual XML format or binary XML format. Possible values are as follows:

- com.ibm.db2.jcc.DB2BaseDataSource.XML_FORMAT_NOT_SET

  Specifies that binary XML format is used if the data server supports it. If the data server does not support binary XML format, textual XML format is used, which is the default.

- com.ibm.db2.jcc.DB2BaseDataSource.XML_FORMAT_TEXTUAL

  Specifies that the XML textual format is used.

- com.ibm.db2.jcc.DB2BaseDataSource.XML_FORMAT_BINARY

  Specifies that the binary XML format is used.

To change the XML format, set the data source property first, then get a new connection, which will pick up your setting. Example 7-3 shows the Java code for setting xmlFormat. Storage and retrieval of binary XML data requires the IBM Data Server Driver for JDBC and SQLJ version 4.9 or later. If you use the DB2 Client, V9.7 Fix Pack 3a and later provides this support.

*Example 7-3 Setting the xmlFormat*

```
DB2BaseDataSource db2ds=null;
...
db2ds.setXmlFormat(com.ibm.db2.jcc.DB2BaseDataSource.XML_FORMAT_BINARY);
con = ((DB2SimpleDataSource)db2ds).getConnection();
```

The format of XML data is transparent to the application. The IBM Data Server Driver for JDBC and SQLJ presents binary XML data to the application only through the XML object interfaces. The user does not see the data in the binary XML format.

Accessing the XML value through SQLXML getSource() and setResult() methods, in which the input or output data is in a non-textual representation, can lead to improved processing performance. The SAX representation is the most efficient way to retrieve data that is in the binary XML format because the data does not undergo extra conversions from binary format to textual format.

# 7.2 The BankStmt application in Java

The BankToCustomerStatement, an ISO 20022 (Universal financial industry message scheme) message is used to inform the account owner, or authorized party, of the entries that are booked to the account, and to provide the owner with balance information about the account at a given time.

The BankToCustomerStatement message is composed of two building blocks:

- Group Header: This building block is mandatory and present once. It contains elements such as Message Identification and CreationDateTime.
- Statement: This building block is mandatory and repetitive. It should be repeated for each account on which a statement is provided. The report contains components such as Balance and Entry.

Chapter 3, "Application scenario" on page 47 presents a scenario to log and store the message for auditing purposes. The diagram in Figure 3-2 on page 51 illustrates the flow between the various code samples presented in this book.

In our Java application, we retrieve, manipulate, and resave those XML documents in DB2, shred the previously saved XML message (see 7.2.5, "Call stored procedure to shred XML" on page 147), and query the message to produce an XML document to send out through an MQ message. The display of this data is generated by combining the XML output with an XSLT file to produce an new HTML or XML display.

The purpose of the Java application is to demonstrate how to use Java with DB2 pureXML and we have chosen to just emphasize the steps and choices made that are related to XML, disregarding irrelevant code and components.

## 7.2.1 Setting up the environment

Before creating the actual Java application, we assume that the necessary environment has been set up:

- Java SDK and IBM Data Server Driver for JDBC and SQLJ are installed on client side.
- The table is defined and the schema is registered on server side.

### Java environment on client

To run the BankStmt application, install an SDK for Java Version 6 or later. To verify the Java version you have, run the following command in your command window:

```
java -version
```

To run the BankStmt application, set up the IBM Data Server Driver for JDBC and SQLJ version 4.9 or later on your client. If you use the DB2 Client, V9.7 Fix Pack 3a and later provides this driver. To verify the driver version you have, run the following command in your command window:

```
java com.ibm.db2.jcc.DB2Jcc -version
```

## Schema registration

We must register XML schemas in the DB2 XML schema repository to be able to validate our XML documents. To register the schema, you can either call the DB2-supplied stored procedures or command-line processor, or through the Java method. Example 7-4 shows how to register the schema in a Java application by calling the registerDB2XmlSchema method.

*Example 7-4 Register XML schema in Java application*

```
String NameSchema = "CAMT053_JAVA";
String XSDFilename = "camt.053.001.02.xsd";
...
//************************************************************************
// Deregister the XML schema if it already exists.
//************************************************************************

try {
   ds.deregisterDB2XmlObject(
               "SYSXSR", NameSchema);
}
catch (SQLException e) {}

System.out.println("deregister complete");

//************************************************************************
//register the XML schema
//************************************************************************

fi[0]= new FileInputStream(XSDFilename);

// Schema Name Qualifiers, always this one, or blank which will default to SYSXSR
xmlSchemaNameQualifiers[0] = "SYSXSR";

// Schema Name
xmlSchemaNames[0] = NameSchema;

//Schema Location: Null means a schema can't be referred by "schemaLocation" in
document.
xmlSchemaLocations[0] = LocationName;

//The actual contents of the schema documents.
xmlSchemaDocuments[0] = new BufferedInputStream(fi[0]);

//Lengths of the actual contents of the schema documents.
xmlSchemaDocumentsLengths[0] = (int) fi[0].getChannel().size();

//Properties of the schema documents. Null means no properties.
xmlSchemaDocumentsProperties[0] = null;

//Lengths of the properties of the whole schema.
xmlSchemaDocumentsPropertiesLengths[0] = 0;

//Properties of the whole schema. Null means no properties.
xmlSchemaProperties = null;

//Lengths of the properties of the whole schema.
```

```
xmlSchemaPropertiesLength = 0;

System.out.println("Register one schema begins...");

ds.registerDB2XmlSchema(
      xmlSchemaNameQualifiers,
      xmlSchemaNames,
      xmlSchemaLocations,
      xmlSchemaDocuments,
      xmlSchemaDocumentsLengths,
      xmlSchemaDocumentsProperties,
      xmlSchemaDocumentsPropertiesLengths,
      xmlSchemaProperties,
      xmlSchemaPropertiesLength,
      false);

System.out.println("Register schema completed.");
```

### DDL for tables in BankStmt application

The BankToCustomerStatement message is received from DB2 MQ Listener and stored in the BK_TO_CSTMR_STMT table.

The DDL for the table is in Example 7-5. The values of MSG_ID and MSG_CRE_DT_TM columns come from the XML document itself. The corresponding XML element name is MsgId (Message Identification), CreDtTm (Creation Date Time) in Group Header of the XML document.

*Example 7-5 DDL for table BK_TO_CSTMR_STMT*

```
CREATE TABLE  "BK_TO_CSTMR_STMT"  (
    "MSG_ID" VARCHAR(35)   WITH DEFAULT NULL       ,
    "MSG_CRE_DT_TM" TIMESTAMP WITH TIMEZONE WITH DEFAULT NULL,
    "BK_TO_CSTMR_STMT" XML NOT NULL )
IN DATABASE XMLR5DB;
```

The XML documents can be validated either explicitly by using the DSN_XMLVALIDATE built-in function, or you can automate XML schema validation by adding an XML type modifier to an XML column definition. This approach is described in Chapter 5, “Validating XML data” on page 75. In this chapter, we show the validation through DSN_XMLVALIDATE function.

## 7.2.2 Insertion of rows with XML column values

Our scenario, described in Chapter 3, “Application scenario” on page 47, shows the stored procedure inserting the XML document into the BK_TO_CUST_STMT table after validating and shredding out MSG_ID and MSG_CRE_DT_TM from the input parameter.

Because Java provides various XML Parser APIs to parse XML documents, shredding out useful information from XML data and inserting it into DB2 can be easy to do.

Example 7-6 on page 142 shows that we read the XML document from a file, parse it, get the MSG_ID and MSG_CRE_DT_TM by using DOM APIs, insert the data, and use DSN_XMLVALIDATE to validate the XML document during insert.

*Example 7-6 Parsing XML value and inserting into DB2*

```
SQLXML sqlxml1 = con1.createSQLXML();
OutputStream os = sqlxml1.setBinaryStream();
//************************************************************************
// get SQLXML object from file
//************************************************************************
try{
   fis = new FileInputStream(XMLFilename);
   int read;
   while ((read = fis.read ()) != -1) {
      os.write (read);
   }
} catch (FileNotFoundException e) {
   e.printStackTrace();
} catch (IOException e) {
   e.printStackTrace();
}

//************************************************************************
// parse the XML value with a DOM parser
//************************************************************************
try{
   DocumentBuilder parser =
      DocumentBuilderFactory.newInstance().newDocumentBuilder();
   Document result = parser.parse(XMLFilename); 1

   //get MSG_ID
   NodeList nodes = result.getElementsByTagName("MsgId"); 2
   MsgId = nodes.item(0).getFirstChild().getNodeValue(); 3

   //get MSG_CRE_DT_TM
   nodes = result.getElementsByTagName("CreDtTm");
   CreDtTm = nodes.item(0).getFirstChild().getNodeValue();
   //convert XML timestamp format to java format
   CreDtTm = CreDtTm.replace("T", " "); 4
} catch (Exception e) {
   e.printStackTrace();
}

//************************************************************************
// insert XML data, also use DSN_XMLVALIDATE to validate
//************************************************************************
String sql = "INSERT INTO BK_TO_CSTMR_STMT " +
      "VALUES(?,?,DSN_XMLVALIDATE(CAST(? AS XML),'SYSXSR.CAMT053_JAVA'))"; 5
pst = con1.prepareStatement(sql);

pst.setString(1, MsgId);
pst.setString(2, CreDtTm);
pst.setSQLXML(3, sqlxml1); 6

pst.executeUpdate(); 7
```

The numbered steps in Example 7-6 on page 142 are as follows:

1. The parser.parse() method parses the content of the given file as an XML document and returns a new DOM Document object.
2. This step finds the "MsgId" element, by using the getElementsByTagName() method, and returns a node list.
3. Because we are expecting only one node in the nodelist, we simply use item(0) to get the first node in the nodelist, and then use getFirstChild().getNodeValue() method to get the value of text node.
4. The XML datetime format uses the letter T as a separator indicating that time-of-day (such as 2010-10-18T17:00:00+01:00) is not compatible with Java datetime format, which requires a space between day and hour (such as 2010-10-18 17:00:00+01:00). This step converts it.
5. Call the DSN_XMLVALIDATE(CAST(? AS XML) function to validate the XML document.
6. Set the parameter to the given java.sql.SQLXML object.
7. Execute the insert. The XML document is inserted as binary XML format (data that is in the Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format), if the data server supports binary XML data.

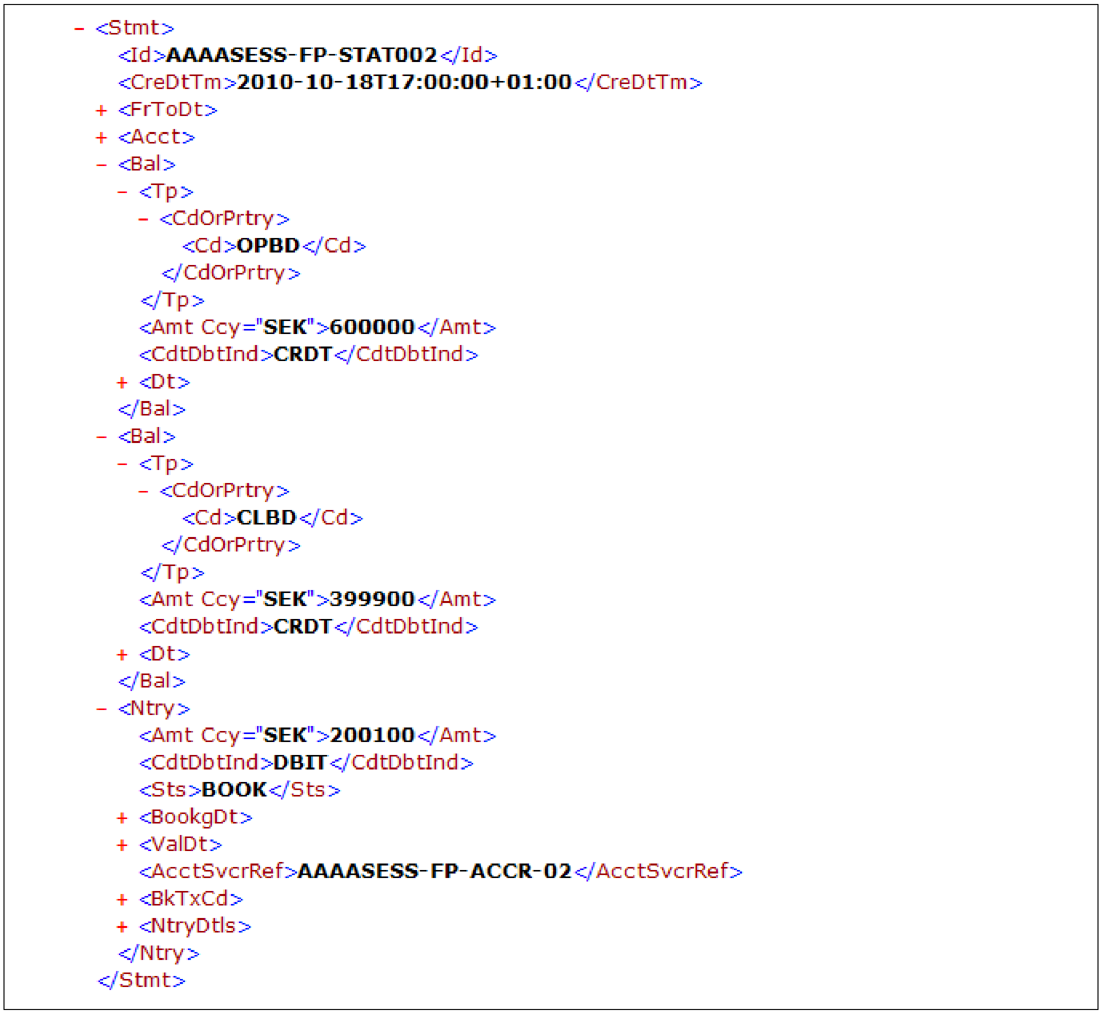
### 7.2.3 Updates of XML columns

You can use the SQL UPDATE statement to update entire documents in an XML column, or update portions of XML documents using the XMLMODIFY function with a basic XPath updating expression.

To update the entire XML documents, you can execute a Java statement or execute a prepareStatement with a setXXX method to set the designated parameter to an XML value.

To update portions of XML documents, use the SQL UPDATE statement with the XMLMODIFY built-in scalar function. The XMLMODIFY function specifies a basic updating expression that you can use to insert nodes, delete nodes, replace nodes, or replace the values of a node in XML documents that are stored in XML columns.

We have inserted a Bank To Customer Statement message, and the statement that reports on booked entries and balances for a cash account is shown as Figure 7-2 on page 144.



```
- <Stmt>
    <Id>AAAASESS-FP-STAT002</Id>
    <CreDtTm>2010-10-18T17:00:00+01:00</CreDtTm>
  + <FrToDt>
  + <Acct>
  - <Bal>
    - <Tp>
      - <CdOrPrtry>
          <Cd>OPBD</Cd>
        </CdOrPrtry>
      </Tp>
      <Amt Ccy="SEK">600000</Amt>
      <CdtDbtInd>CRDT</CdtDbtInd>
    + <Dt>
    </Bal>
  - <Bal>
    - <Tp>
      - <CdOrPrtry>
          <Cd>CLBD</Cd>
        </CdOrPrtry>
      </Tp>
      <Amt Ccy="SEK">399900</Amt>
      <CdtDbtInd>CRDT</CdtDbtInd>
    + <Dt>
    </Bal>
  - <Ntry>
      <Amt Ccy="SEK">200100</Amt>
      <CdtDbtInd>DBIT</CdtDbtInd>
      <Sts>BOOK</Sts>
    + <BookgDt>
    + <ValDt>
      <AcctSvcrRef>AAAASESS-FP-ACCR-02</AcctSvcrRef>
    + <BkTxCd>
    + <NtryDtls>
    </Ntry>
  </Stmt>
```

*Figure 7-2 Bank To Customer Statement example*

The first balance (Bal element with the Cd code OPBD) shows that the book balance of the account at the beginning of the account reporting period is 600000 Swedish krona. There is one entry (Ntry) in the statement, and the code of CdtDbtInd is DBIT which indicates the balance is a a debit balance, so the operation is a decrease. The Amount is 200100 Swedish krona. The second balance (Bal element with the Cd code CLBD) shows the balance of the account at the end of the pre-agreed account reporting period. It is the sum of the opening booked balance at the beginning of the period and all entries booked to the account during the pre-agreed account reporting period, so the amount is 399900 Swedish krona.

To modify or correct the message, add a new entry which is a credit balance increase of 100100 Swedish krona. We perform the following modification:

1. Append a new entry (Ntry) after the first entry, record the credit balance of 100100 Swedish krona.
2. Modify the amount of ClosingBooked (CLBD) balance to 500000 Swedish krona

Example 7-7 shows how to modify an XML document, insert nodes, and replace the values of a node.

*Example 7-7 Modifying an XML document*

```
//***********************************************************************
// update XML document, add a new entry as the last entry
//***********************************************************************

String sql = " UPDATE BK_TO_CSTMR_STMT "+
"SET BK_TO_CSTMR_STMT = XMLMODIFY ( "+ 1
"'declare default element namespace " +
"\"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02\"; "+
"insert nodes $newentry/newNtry/Ntry "+ 2
"after /Document/BkToCstmrStmt/Stmt/Ntry[fn:last()]', "+ 3
"XMLPARSE(DOCUMENT " +
"        '<newNtry xmlns=\"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02\"> "+
"         <Ntry> "+
"           <Amt Ccy=\"SEK\">100100</Amt> "+
"           <CdtDbtInd>CRDT</CdtDbtInd> "+
"           <Sts>BOOK</Sts> "+
"           <BkTxCd> "+
"              <Domn> "+
"                 <Cd>PAYM</Cd> "+
"                 <Fmly> "+
"                    <Cd>0001</Cd> "+
"                    <SubFmlyCd>0005</SubFmlyCd> "+
"                 </Fmly> "+
"              </Domn> "+
"           </BkTxCd> "+
"        </Ntry> "+
"        </newNtry>') as \"newentry\") "+
"WHERE MSG_ID=? ";

pst = con1.prepareStatement(sql);
pst.setString(1, "AAAASESS-FP-STAT002");
pst.executeUpdate();

//***********************************************************************
// update XML document, modify the amount of ClosingBooked(CLBD) balance
//***********************************************************************

sql = " UPDATE BK_TO_CSTMR_STMT "+
"SET BK_TO_CSTMR_STMT = XMLMODIFY ( "+
"'declare default element namespace " +
"\"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02\"; "+
" replace value of node " + 4
"/Document/BkToCstmrStmt/Bal[Tp/CdOrPrtry/Cd=\"CLBD\"]/Amt" + 5
" with \"500000\"') "+
"WHERE MSG_ID=? ";

pst = con1.prepareStatement(sql);
pst.setString(1, "AAAASESS-FP-STAT002");
pst.executeUpdate();
```

The numbered steps in Example 7-7 on page 145 are as follows:

**1.** Invoke XMLMODIFY function to insert or replace XML nodes.

**2.** This UPDATE is to insert a new Ntry node.,

**3.** The `fn:last` function returns the number of Ntry nodes. The last Ntry node of the document is `/.../Ntry[fn:last()]`. The new Ntry node will be inserted after the last Ntry node.

**4.** This UPDATE is to replace value of a node.

**5.** Replaces the Amt value when `.../Cd` is "`CLBD`".

## 7.2.4 Retrieving XML data

Use one of the following ways to retrieve XML data:

- Use the `ResultSet.getSQLXML` method to retrieve the data. Then, use a `SQLXML.getXXX` method to retrieve the data into a compatible output data type. This technique requires JDBC 4.0 or later.
- Use a `ResultSet.getXXX` method other than `ResultSet.getObject` to retrieve the data into a compatible data type.
- If you use JCC3 driver that does not support JDBC 4.0, you may use the `ResultSet.getObject` method to retrieve the data, and then cast it to the DB2Xml type and assign it to a DB2Xml object. Use a `DB2Xml.getDB2XXX` or `DB2Xml.getDB2XmlXXX` method to retrieve the data into a compatible output data type.

To retrieve the entire XML document or partial XML document, use the XMLQUERY function. You can retrieve data from XML columns in a table as binary XML data (data that is in the Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format), if the data server supports binary XML data. Example 7-8 shows how to retrieve the entire or partial XML document.

*Example 7-8 Retrieving the entire or partial XML document*

```
//************************************************************************
// retrieve XML document to a SQLXML object
//************************************************************************
String sql = "SELECT BK_TO_CSTMR_STMT FROM BK_TO_CSTMR_STMT WHERE MSG_ID = ?";
pst = con1.prepareStatement(sql);
pst.setString(1, "AAAASESS-FP-STAT002");
ResultSet rs=pst.executeQuery();

while (rs.next()) {
   sqlxml=rs.getSQLXML(1);
   actualResults.println(sqlxml.getString());
}

//************************************************************************
// retrieve the XML nodes by using XMLQUERY to a SQLXML object
//************************************************************************
sql = " SELECT XMLQUERY(" +
      "'declare default element namespace " +
      "\"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02\"; " +
      "/Document/BkToCstmrStmt/Stmt/Bal/Amt' " +
      "passing BK_TO_CSTMR_STMT ) " +
      "FROM BK_TO_CSTMR_STMT " +
      "WHERE MSG_ID = ?";
pst = con1.prepareStatement(sql);
pst.setString(1, "AAAASESS-FP-STAT002");
rs=pst.executeQuery();

while (rs.next()) {
   sqlxml=rs.getSQLXML(1);
   actualResults.println(sqlxml.getString());
}
```

### 7.2.5 Call stored procedure to shred XML

The BankStmt application continues to shred the BankToCustomerStatement message from BK_TO_CSTMR_STMT table into a STMT table, which demonstrates a simple hybrid design, and how to populate using the INSERT from SELECT with XMLTABLE. We perform the shredding in a native stored procedure. Our java application retrieves the XML message and calls the stored procedure with the XML value as parameter.

Example 7-9 shows the definition for the STMT table, each row contains one statement (Stmt building block) of the BankToCustomerStatement message. For example "STMT_ID" column corresponds to the 'Id' node under Stmt element, and "STMT_XML" column will be the sub document (the Stmt building block) of the BankToCustomerStatement message.

*Example 7-9 DDL for STMT table*

```
CREATE TABLE  "STMT"  (
    "STMT_ID" VARCHAR(35) NOT NULL ,
    "MSG_ID" VARCHAR(35) ,
    "MSG_CRE_DT_TM" TIMESTAMP ,
    "ELECTRNC_SEQ_NB" BIGINT ,
    "LGL_SEQ_NB" BIGINT ,
    "CRE_DT_TM" TIMESTAMP NOT NULL ,
    "FR_DT_TM" TIMESTAMP ,
    "TO_DT_TM" TIMESTAMP ,
    "RPTG_SRC_CD" CHAR(4) NOT NULL ,
    "RPTG_SRC_PRTRY" VARCHAR(35) NOT NULL ,
    "ADDTL_INF" VARCHAR(140) NOT NULL,
    "STMT_XML" XML NOT NULL )
   IN DATABASE XMLR5DB ;
```

Example 7-10 shows our code for creating the SQL procedure in Java.

*Example 7-10 Creating a SQLstored procedure*

```
stmt.executeUpdate("" +
   "CREATE PROCEDURE MYSP(IN parm1 XML,OUT parm2 XML)" + 1
   "LANGUAGE SQL          " +
   "APPLICATION ENCODING SCHEME UNICODE" +
   "DISABLE DEBUG MODE " +
   "BEGIN                   " +
   "DECLARE var1 XML;  " +
   "SET var1 = parm1;  " +
   "INSERT INTO STMT(  " + 2
   "  STMT_ID,          " +
   "  MSG_ID,              " +
   "  MSG_CRE_DT_TM,       " +
   "  ELECTRNC_SEQ_NB, " +
   "  LGL_SEQ_NB,          " +
   "  CRE_DT_TM,           " +
   "  FR_DT_TM,            " +
   "  TO_DT_TM,            " +
   "  RPTG_SRC_CD,         " +
   "  RPTG_SRC_PRTRY,      " +
   "  ADDTL_INF,           " +
   "  STMT_XML             " +
   ")                        " +
   "SELECT T.STMT_ID,  " +
   "     T.MSG_ID,         " +
   "     T.MSG_CRE_DT_TM," +
   "     T.ELECTRNC_SEQ_NB," +
   "     T.LGL_SEQ_NB,     " +
   "     T.CRE_DT_TM,      " +
   "     T.FR_DT_TM,       " +
   "     T.TO_DT_TM,       " +
   "     COALESCE(T.RPTG_SRC_CD,'') AS PRTG_SRC_CD," + 3
   "     COALESCE(T.RPTG_SRC_PRTRY,'') AS RPTG_SRC_PRTRY," +
   "     COALESCE(T.ADDTL_INF,'') AS ADDTL_INF," +
   "     XMLDOCUMENT(T.STMT_XML)" + 4
   "FROM XMLTABLE(         " +
   "  XMLNAMESPACES(DEFAULT " +
   "      'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02')," +
   "  '$var1/Document/BkToCstmrStmt/Stmt'" +
   "  PASSING var1 as \"var1\"" +
   "  COLUMNS STMT_ID VARCHAR(35) PATH 'Id'," +
   "        MSG_ID VARCHAR(35)  PATH '../GrpHdr/MsgId'," +
   "        MSG_CRE_DT_TM TIMESTAMP  PATH '../GrpHdr/CreDtTm'," +
   "        ELECTRNC_SEQ_NB BIGINT  PATH 'ElctrncSeqNb'," +
   "        LGL_SEQ_NB BIGINT       PATH 'LglSeqNb'," +
   "        CRE_DT_TM TIMESTAMP PATH 'CreDtTm'," +
   "        FR_DT_TM   TIMESTAMP PATH    'FrToDt/FrDtTm'," +
   "        TO_DT_TM   TIMESTAMP PATH    'FrToDt/ToDtTm'," +
   "        RPTG_SRC_CD    CHAR(4) PATH  'RptgSrc/Cd'," +
   "        RPTG_SRC_PRTRY VARCHAR(35) PATH 'RptgSrc/Prtry'," +
   "        ADDTL_INF   VARCHAR(144) PATH   'AddtlStmtInf'," +
   "        STMT_XML   XML     PATH     '.'" +
   "  )AS T;                 " +
   //************************************************************************
   // For the output parameter, You can do some more XML operations,
   // here we just simply set the output parameter the same as input
   //************************************************************************
   "SET parm2 = var1;  " +
   "END
);
```

The numbered steps in Example 7-10 on page 147 are as follows:

**1.** In DB2 10, we can use XML as the data type for a parameter of a native SQL procedure, and an XML SQL variable declared within the procedure.

**2.** We use INSERT from SELECT with the XMLTABLE function for the shredding. If the BankToCustomerStatement message includes multiple statements, we are able to divide one message into multiple statements (one rows for each statement) in the STMT table by using the XMLTABLE function.

**3.** To insert a nullable value into a NOT NULL column, we use the COALESCE function.

**4.** When inserting the XML value from XMLTABLE, we use the XMLDOCUMENT function to return a constructed XML value.

To call the stored procedures in Java, invoke methods in the CallableStatement class. The following basic steps call stored procedures, using standard CallableStatement methods:

1. Invoke the `Connection.prepareCall` method with the CALL statement as its argument to create a CallableStatement object.
2. Invoke the `CallableStatement.setXXX` methods to pass values to the input parameters (parameters that are defined as IN or INOUT in the CREATE PROCEDURE statement).
3. Invoke the `CallableStatement.registerOutParameter` method to register parameters that are defined as OUT in the CREATE PROCEDURE statement with specific data types.
4. Invoke the `CallableStatement.executeXXX` methods to call the stored procedure.
5. Invoke the `CallableStatement.getXXX` methods to retrieve values from the OUT parameters or INOUT parameters.

The sample code to prepare, call the stored procedure, and retrieve the XML data from OUT parameters is shown in Example 7-11.

*Example 7-11 Handling the SQL stored procedure*

```
String sql = "CALL MYSP(?,?)";
CallableStatement cstmt = con1.prepareCall(sql);

//initialize the parms
SQLXML xml1 = con1.createSQLXML();

stmt = con1.createStatement();
sql = "SELECT BK_TO_CSTMR_STMT FROM BK_TO_CSTMR_STMT " +
      "WHERE MSG_ID='AAAASESS-FP-STAT003-4'";
ResultSet rs = stmt.executeQuery(sql);
if(rs.next())
   xml1=rs.getSQLXML(1);

actualResults.println("value of input XML:");
actualResults.println(xml1.getString());

cstmt.setSQLXML(1, xml1);
cstmt.registerOutParameter(2, java.sql.Types.SQLXML);
cstmt.execute();

xml1 = cstmt.getSQLXML(2);
actualResults.println("value of output XML:");
actualResults.println(xml1.getString());

cstmt.close();
```

### 7.2.6 XSLT to transform XML document

With XSLT (see 1.2.4, “Extensible Stylesheet Language” on page 12), you can transform an XML document into another XML document, or another type of document that is recognized by a browser, such as HTML and XHTML. With XSLT, you can add or remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display.

Java API for XML Processing (JAXP) provides interfaces in the `javax.xml.transform` package, so that applications can invoke an XSLT transformation.

In the BankStmt application, we want to generate a report that lists all entries that have an amount greater than 100,000 SEK. The expected output is shown in Example 7-12. In the output, we list the account ID, amount, date and time, and account servicer reference of the entry.

*Example 7-12 Expecting output after transform*

```
<?xml version="1.0" encoding="UTF-8"?>
<Result xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02"
   xmlns:iso20022="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02">
   <Acct Id="50000000054910000005">
      <Ntry>
         <Amt Ccy="SEK">150000</Amt>
         <DtTm>2010-10-19T10:15:00+01:00</DtTm>
         <AcctSvcrRef>AAAASESS-FP-ACCR-03</AcctSvcrRef>
      </Ntry>
   </Acct>
   <Acct Id="50000000054910000006">
      <Ntry>
         <Amt Ccy="SEK">200000</Amt>
         <DtTm>2010-10-20T10:15:00+01:00</DtTm>
         <AcctSvcrRef>AAAASESS-FP-ACCR-05</AcctSvcrRef>
      </Ntry>
   </Acct>
</Result>
```

We use an XSLT file to transform the original BankToCustomerStatement message to a new XML document. The simple XSLT file is shown in Example 7-13.

*Example 7-13 XSLT example to transform from XML to XML*

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:iso20022="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02"
                xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02">
<!-- root template -->
<xsl:template match="/">
<Result>
   <xsl:for-each 1
select="/iso20022:Document/iso20022:BkToCstmrStmt/iso20022:Stmt/iso20022:Ntry">
      <xsl:apply-templates select="."/> 2
   </xsl:for-each>
</Result>
</xsl:template>
<!-- Ntry template -->
```

```
<xsl:template match="iso20022:Ntry">
    <xsl:if test="iso20022:Amt>100000"> 3
   <Acct Id="{../iso20022:Acct/iso20022:Id/iso20022:Othr/iso20022:Id}"> 4
      <Ntry>
         <Amt Ccy="{iso20022:Amt/@Ccy}">
            <xsl:value-of select="iso20022:Amt"/>
         </Amt>
         <DtTm>
            <xsl:value-of select="iso20022:BookgDt/iso20022:DtTm"/>
         </DtTm>
         <AcctSvcrRef>
            <xsl:value-of select="iso20022:AcctSvcrRef"/>
         </AcctSvcrRef>
      </Ntry>
    </Acct>
    </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

The numbered steps in Example 7-13 on page 150 are as follows:

1. The xsl:for-each element repeats for each Ntry node, which is selected by the XPath:

   `/iso20022:Document/iso20022:BkToCstmrStmt/iso20022:Stmt/iso20022:Ntry`

2. For each Ntry node, applies Ntry template to do more transformations.

3. In Ntry template, the xsl:if element tests whether the amount is greater than 100000.

4. Constructs the expecting Acct element with attribute `Id`.

For more information about XSLT standard, go to the following web address:

http://www.w3.org/TR/xslt

The Java application reads the BankToCustomerStatement message stored in the BK_TO_CSTMR_STMT table, transforms it into a new XML document based on the XSLT file, and stores it in a file. The code snippet is shown in Example 7-14.

*Example 7-14 Java application to transform XML document*

```
//retrieve xml document from BK_TO_CSTMR_STMT
String sql = "SELECT BK_TO_CSTMR_STMT FROM BK_TO_CSTMR_STMT " +
      "WHERE MSG_ID='AAAASESS-FP-STAT003-4'";
ResultSet rs = stmt.executeQuery(sql);
if(rs.next())
   xml1=rs.getSQLXML(1);

// JAXP reads data using the Source interface
Source xmlSource = new StreamSource(xml1.getBinaryStream());
Source xsltSource = new StreamSource(xsltFile);

// the factory pattern supports different XSLT processors
TransformerFactory transFact =
TransformerFactory.newInstance();
Transformer trans = transFact.newTransformer(xsltSource);

//transform and put the new XML document into a file
Result result = new StreamResult(xmloutputFile);
trans.transform(xmlSource, result);
```

Transforming an XML document to an HTML display is similar. The only difference is that we must transform the HTML document with a standard tag and attribute name, as shown in Example 7-15.

*Example 7-15   XSLT to transform from XML to HTML*

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:iso20022="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02"
>
<!-- root template -->
<xsl:template match="/">
<html> 1
   <body>
      <table border="1" cellspacing="0"> 2
         <tr>
            <th>Acct(Id)</th> 3
            <th>Amt(Ccy="SEK")</th>
            <th>DtTm</th>
            <th>AcctSvcrRef</th>
         </tr>
         <xsl:for-each select="//iso20022:Ntry"> 4
            <xsl:apply-templates select="."/>
         </xsl:for-each>
      </table>
   </body>
</html>
</xsl:template>
<!-- Ntry template -->
<xsl:template match="iso20022:Ntry">
    <xsl:if test="iso20022:Amt>100000">
    <tr>
      <td>
         <xsl:value-of
select="../iso20022:Acct/iso20022:Id/iso20022:Othr/iso20022:Id"/>
      </td>
      <td>
         <xsl:value-of select="iso20022:Amt"/>
      </td>
      <td>
         <xsl:value-of select="iso20022:BookgDt/iso20022:DtTm"/>
      </td>
      <td>
         <xsl:value-of select="iso20022:AcctSvcrRef"/>
      </td>
    </tr>
    </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

The numbered steps in Example 7-15 on page 152 are as follows:

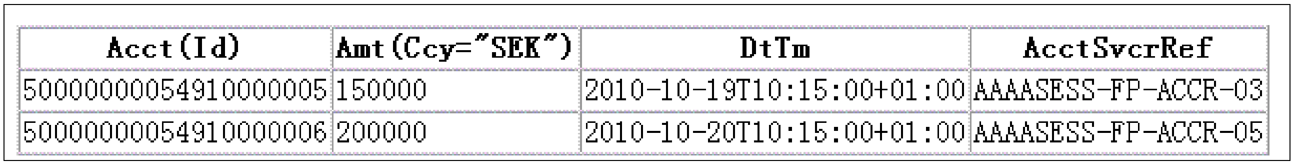
1. Start building a HTML document.

2. Create a simple table in the HTML document.

3. Define the header information using `<th>` tag.

4. For each Ntry, apply Ntry template to convert it into one row in the table.

The display of the HTML document is in Figure 7-3.



| Acct(Id) | Amt(Ccy="SEK") | DtTm | AcctSvcrRef |
|---|---|---|---|
| 50000000054910000005 | 150000 | 2010-10-19T10:15:00+01:00 | AAAASESS-FP-ACCR-03 |
| 50000000054910000006 | 200000 | 2010-10-20T10:15:00+01:00 | AAAASESS-FP-ACCR-05 |

*Figure 7-3 HTML output after XSLT*

## 7.2.7 Java interface to MQ

WebSphere MQ can help you more easily exchange information among platforms, integrating new and existing business applications. The BankStmt application scenario (XML message logging and auditing, in Chapter 3, “Application scenario” on page 47) describes a common situation, where XML messages are flowing over a WebSphere MQ. Chapter 6, “DB2 SQL/XML programming” on page 89 describes how to capture these messages and store them in DB2 pureXML for auditing purposes. In this section, after retrieving the messages from DB2 and performing some transformations using XSLT, our Java application puts the generated XML document in the WebSphere MQ.

To add a message to a queue, you must connect to the queue manager first, which provides queuing services to the applications. Then, you can open a queue and put the message into it. Figure 7-4 on page 154 show the process.

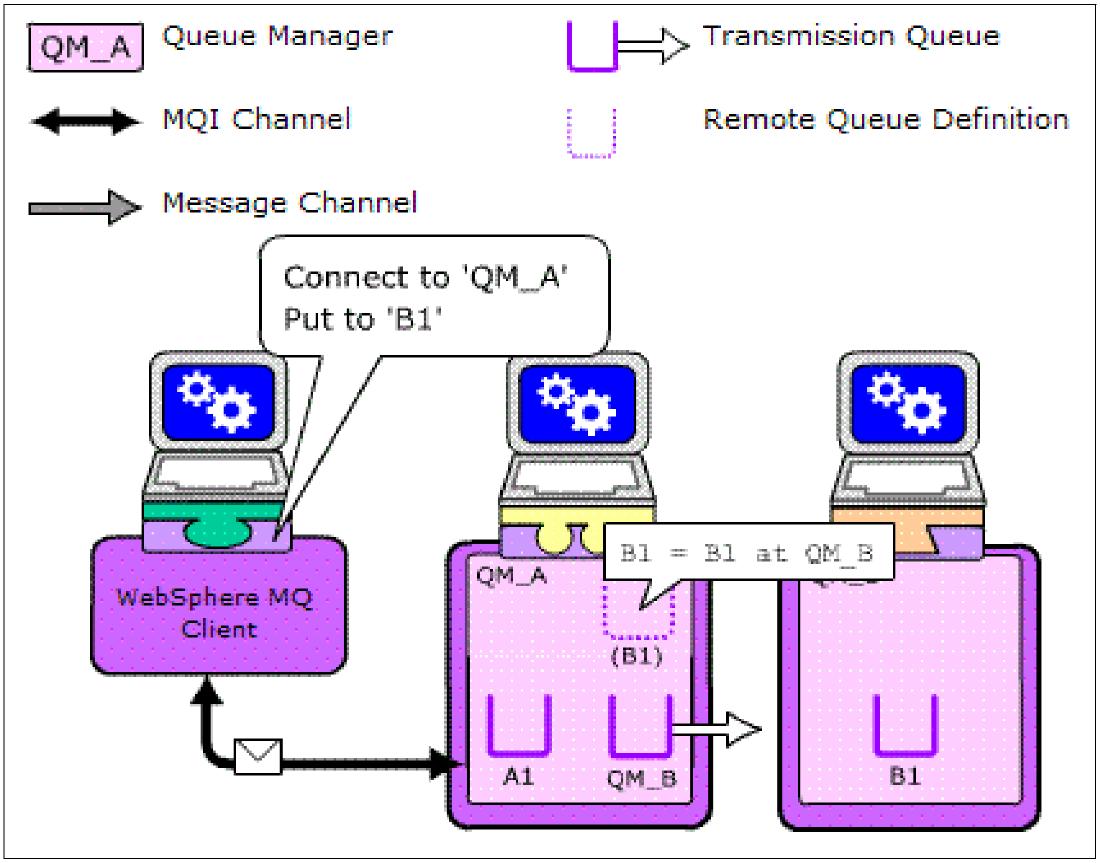


*Figure 7-4 Put a message into a queue*

WebSphere MQ classes for Java Message Service (also referred to as WebSphere MQ JMS) is a set of Java classes that implement Oracle JMS (Sun) interfaces to enable JMS programs to access WebSphere MQ systems.

Using WebSphere MQ JMS as the API to write WebSphere MQ applications has a number of benefits. Several advantages derive from JMS being an open standard with multiple implementations. Other advantages are from additional features that are present in WebSphere MQ JMS, but not in WebSphere MQ base Java, such as asynchronous message delivery, message selectors, and support for publish/subscribe messaging. WebSphere MQ JMS APIs can be used to connect to MQ on multiple platforms.

The example code to put the message into a queue is shown in Example 7-16 on page 155.

*Example 7-16 Java example to put a message into a queue*

```
//************************************************************************
// Before starting the example, you need input the following information:
// IP address(ipaddress), port number(port), channel name(channel)
// queue manager name(queueMgr) and queue name(queuename)
//************************************************************************

//************************************************************************
// Maps the specified key to the specified value into properties
//************************************************************************
properties = new Hashtable();

properties.put("hostname", ipaddress);
properties.put("port", new Integer(Integer.parseInt(port)));
properties.put("channel", channel);

//************************************************************************
// Connect to MQ
//************************************************************************
try {
   queueManager = new MQQueueManager(queueMgr, properties);
}
catch (java.lang.NoClassDefFoundError e) {
   ...
}

//************************************************************************
// Open a queue
//************************************************************************
MQQueue system_default_local_queue=null;

try {
   system_default_local_queue = queueManager.accessQueue(queuename,
      MQConstants.MQOO_INPUT_AS_Q_DEF | MQConstants.MQOO_OUTPUT,
      null, null, null);
}
catch (MQException ex) {
   ...
}

//************************************************************************
// Put a message into the queue
//************************************************************************
MQMessage put_msg = new MQMessage();
put_msg.characterSet = 1208;
try {
   put_msg.writeUTF(xmlmessage);
}
catch (Exception ex) {
   ...
}

try {
   system_default_local_queue.put(put_msg, new MQPutMessageOptions());
}
catch (MQException ex) {
   ...
}
```

All Java examples are available as additional material, described in Appendix B, “Additional material” on page 277.

The sample code includes the following classes:

- RegisterSchema.java

  This class registers the XML schema to the DB2 databases

- InsertXML.java

  This class validate and insert XML data into db2 table.

- UpdateXML.java

  This class demo partial updates of XML documents

- SelectXML.java

  This class demo the retrieving entire or partial XML document to a SQLXML object

- XMLProcedure.java

  This class creates SQL stored procedure with XML as parameter to shred XML document, then call the SQL stored procedure from Java

- TransformXML.java

  This class transform XML document retrieved into an new XML or HTML document

- SendMQMessage.java

  This class send a XML message generated by XSLT to WebSphere MQ

The data we used is as follows:

- Schema of bank to customer statement message

  ```
  camt.053.001.02.xsd
  ```

- XML file of bank to customer statement message

  ```
  camt.053.001.02.xml
  camt.053.001.03.xml
  camt.053.001.04.xml
  ```

- XSLT file to transform XML message

  ```
  camt.053.001.04.xsl
  camt.053.001.05.xsl
  ```

- XML document send to WebSphere MQ

  ```
  xmloutput.xml
  ```

8

# Using XML with COBOL

With Enterprise COBOL for z/OS V4.1, you can integrate COBOL and web-based business processes in web services, XML, Java, and COBOL applications. This interoperability can help you capitalize on existing IT investment and incorporate new, web-based applications as part of your organization’s infrastructure.

In this chapter, we demonstrate the COBOL support for the DB2 pureXML format by implementing a small COBOL application based on the BankToCustomerStatement message of the ISO 20022 standard presented in Chapter 3, “Application scenario” on page 47.

We also briefly cover native COBOL facilities for XML and how they complement those of DB2 pureXML.

This chapter contains the following topics:

- XML representation in COBOL
- The BankStmt application in COBOL
- COBOL functions for manipulating XML

The complete source code, DDL, XML schema, and scripts that are used in the application are made available for download, as described in Appendix B, “Additional material” on page 277.

© Copyright IBM Corp. 2011. All rights reserved.

# 8.1 XML representation in COBOL

COBOL offers options for working with XML data. Most important, support is available for the pureXML data type in DB2 in several variations, although in several cases ordinary binary or character-based types might also be used. In addition, the file reference variable is applicable to LOBs and XML.

Because XML is always stored in Unicode, pay special attention to which code pages are used in the application and how to avoid data conversion.

We use the DB2 precompiler throughout this chapter. Experiences might vary slightly if using the coprocessor.

## 8.1.1 XML host variables in COBOL

In DB2, the data type XML is a basic data type with its own representation and associated simple functions. You can insert, modify, and retrieve data as XML. In COBOL, the XML data type always builds on one of the existing LOB formats. XML variable declarations built on the basic LOB types are shown in Example 8-1.

*Example 8-1 XML host variables in COBOL*

```
01  DOC-AS-CLOB   IS SQL TYPE IS XML AS CLOB(100K).
01  DOC-AS-BLOB   IS SQL TYPE IS XML AS BLOB(100K).
01  DOC-AS-DBCLOB IS SQL TYPE IS XML AS DBCLOB(100K).
```

These declarations are translated by the precompiler, as shown in Example 8-2.

*Example 8-2 XML host variables after transformation by the DB2 pre-compiler*

```
01  DOC-AS-CLOB.
  49 DOC-AS-CLOB-LENGTH          PIC S9(9) COMP-5.
  49 DOC-AS-CLOB-DATA            PIC X(102400).
01  DOC-AS-BLOB.
  49 DOC-AS-BLOB-LENGTH          PIC S9(9) COMP-5.
  49 DOC-AS-BLOB-DATA            PIC X(102400).
01  DOC-AS-DBCLOB.
  49 DOC-AS-DBCLOB-LENGTH        PIC S9(9) COMP-5.
  49 DOC-AS-DBCLOB-DATA          PIC G(102400) DISPLAY-1.
```

Whether you want to use CLOBs, DBCLOBs, or BLOBs as the base format depends on how and whether you want to manipulate the contents of the XML file in the application.

One significant difference between the base format of BLOB compared to that of CLOB (or DBCLOB) is the encoding. XML is always stored in UTF-8 Unicode; the COBOL application can work in EBCDIC or UTF-16 Unicode. Therefore, data conversion and data encoding is always an issue you must consider.

### Data encoding

The character-based formats are referred to as *externally encoded*; the binary-based formats are referred to as *internally encoded*. A variable with subtype FOR BIT DATA is also considered internally encoded.

Externally encoded means that the code page is derived from the CCSID of the host variable, if one is specified, or else from the application code page. Internally encoded means that the code page is derived from the data itself.

An XML document might or might not contain an encoding declaration, regardless of whether it is internally or externally encoded. It is placed as an attribute of the XML declaration as Example 8-3 shows.

*Example 8-3 XML declaration with encoding declaration as an attribute*

```
<?xml version="1.0" encoding="IBM037"?>
```

However, the significance of the encoding declaration differs depending on the format. For internally encoded documents, the encoding declaration is a factor in deciding the code page of the XML document, together with the Unicode byte order mark (BOM) and the remaining XML declaration. The BOM is an optional byte sequence that precedes the XML document denoting the byte order of the following text. It is used for Unicode documents only.

The exact algorithm for determining the code page of an internally encoded XML document is as follows:

- If the document contains a BOM, the BOM decides the endianess. Possible outcomes are UTF-16, UTF-32, big endian, or little endian.
- If no BOM is present, and the XML document contains an encoding declaration, the encoding declaration decides the code page. Possible outcome is any valid code page, it does not have to be UTF-n encoding.
- If no BOM and no encoding declaration is present, and the document contains an XML declaration, the code page is UTF-8 if the document is written in single-byte ASCII, UTF-16 if the document is double-byte ASCII.
- If no BOM and no XML declaration is present, the code page is UTF-8.

**Note:** If there is a mismatch in endianess between BOM and the UTF-16 XML document, the insert or update fails.

For externally encoded documents, any encoding declarations are ignored by DB2. Instead the code page is determined as with any character data. The default is the application code page, but you may override it in the SQLDA, or by an explicit variable declaration as shown in Example 8-4. This way is probably the easiest, but it requires that the SQLDA is not in use for the variables in question.

*Example 8-4 Explicit declaration of variable CCSID*

```
01 XMLVAR USAGE IS SQL TYPE IS XML AS CLOB
      EXEC SQL DECLARE :XMLVAR VARIABLE CCSID 277 END-EXEC.
```

We conclude that the internally encoded variables and externally encoded variables give different results when used for inserting XML documents into DB2. This information is summarized in Table 8-1 on page 160; the effect of inserting various XML documents using either XML AS BLOB or XML AS CLOB is shown.

We assume that the XML AS CLOB variable has an EBCDIC CCSID associated with it, either through explicit declaration or from the application code page. Also assumed is that the EBCDIC code page referred to is the same through the table. The XML AS BLOB, of course, has no associated CCSID.

*Table 8-1 Insert an XML document with separate host variable types*

| Document encoding | XML AS CLOB, CCSID is EBCDIC | XML AS BLOB |
|---|---|---|
| EBCDIC document with EBCDIC encoding declaration | Insert successful | Insert successful |
| EBCDIC document with UTF-8 encoding declaration | Insert successful | SQL error -20398 |
| UTF-8 document with EBCDIC encoding declaration | SQL error -20398 | SQL error -20398 |
| UTF-8 document with UTF-8 declaration | SQL error -20398 | Insert successful |

Regardless of the encoding format that is used to enter XML data into DB2, ensure that the encoding declaration is accurate, because data might later be transmitted to other components or applications relying on the encoding declaration. If the encoding is not accurate, errors might occur.

For details about how DB2 handles mixed data in a character string, go to the following address:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db2z10.doc.sqlref/db2z_mixeddatainchar.htm

## Avoiding data conversion

In general, avoid data conversion because it is both costly in terms of CPU usage, and has the potential for data loss. Data loss can occur if the resulting code page does not hold all the code points present in the original document.

When an XML document is inserted into DB2, data conversion always takes place if the code page of the XML document is anything but UTF-8. This situation occurs whether the variable holding the XML document is internally or externally encoded. Therefore, for the simple case of inserting an XML document in EBCDIC from a COBOL application, data conversion cannot be avoided even if using a BLOB as the base for the XML column.

However, when retrieving an XML document from DB2, data conversion occurs only if using an externally encoded format for the target variable, and if the associated code page differs from UTF-8. Therefore, although here data conversion is avoided when using a binary format, you then have the data in Unicode, which might or might not be what you want.

For COBOL applications that work with XML documents in an EBCDIC code page, either because they are received and transmitted to other applications in that format, or because they are created and manipulated entirely in COBOL, a good choice is to use externally encoded variables because that ensures that the necessary data conversions are performed.

However, if the COBOL application receives the XML data from, or transmits XML data to an application using another code page, for example through WebSphere MQ, there is the risk of performing an unnecessary interim data conversion.

In this case, using an internally encoded format might be preferable if the external application either uses Unicode, or is able to perform the conversion from Unicode to its own code page. The other option is using external encoding with the CCSID of the external application. Both these choices eliminate the need for an interim conversion of the data, but both require that the COBOL application can handle the XML data in the other application’s code page.

Figure 8-1 shows how the interim code conversion can be avoided by using an internally encoded variable for the XML data.

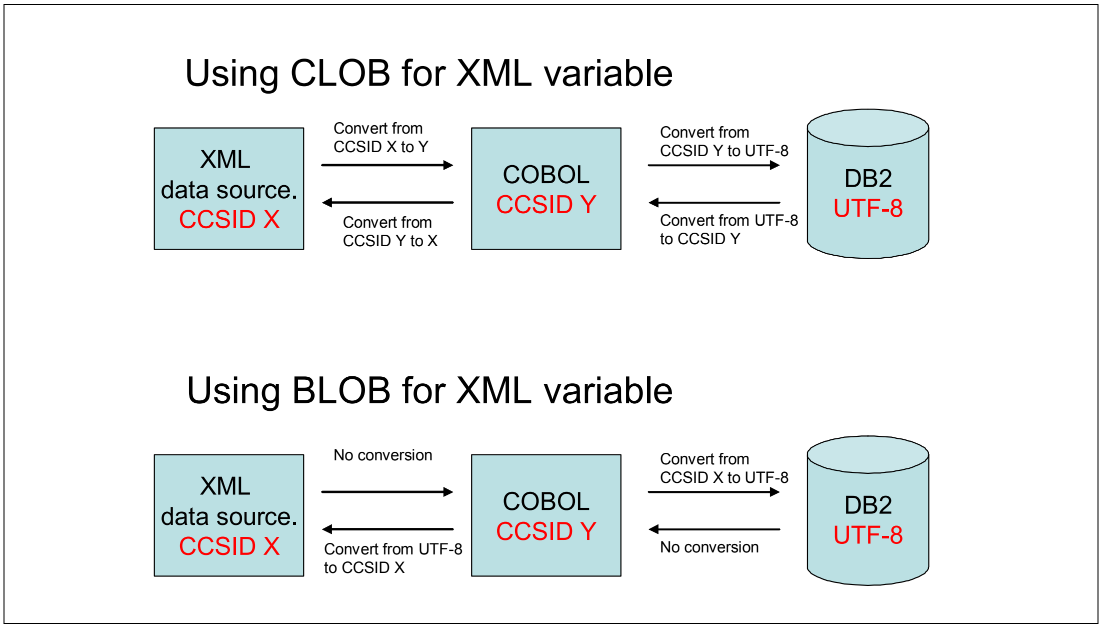


*Figure 8-1 Data conversion in a three-layer structure using CLOBs or BLOBs*

## 8.1.2 Using non-XML variables for XML data

In general, use XML variables for XML data. However in certain cases, consider other options that might be more practical. One such case might be when developing COBOL stored procedures. At present, the XML data type is not supported for external stored procedures.

A possibility is to use non-XML variables for inserting and updating, and for retrieving XML documents. Possible data types are CLOB, BLOB, DBCLOB, CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, BINARY and VARBINARY.

You can either let DB2 handle the conversion implicitly, or explicitly call the conversion functions. XMLSERIALIZE converts the host variable type to XML whereas XMLPARSE converts XML to a host variable type. A more efficient way is to let DB2 perform the conversion implicitly when possible.

In more complex SQL/XML, calls, for example XMLMODIFY, substituting the XML type with other host variable types is not possible. In this case, performing an explicit conversion by calling XMLPARSE or XMLSERIALIZE might be necessary.

**Note:** Always use XMLSERIALIZE WITHOUT XMLDECLARATION with an externally encoded host variable.

The XML declaration resulting from an explicit serialization always contains `encoding="UTF-8"` because the conversion is performed after document retrieval. This setting does not match the contents of the document if any data conversion is performed and may result in application errors.

For implicit serialization, the XML declaration always matches the document contents.

### 8.1.3 Using file reference variables for efficient insert and retrieval

DB2 offers the use of file reference variables for XML data in the same manner as for LOBs. *File reference variables* are variables that point to documents in the file system so that you can use them to refer to the documents without reading the contents of the documents into the memory of the application.

They can be used for efficient insertion of XML documents that you receive from outside the application (if you do not need to manipulate the contents before inserting), and for retrieval of documents that you want to pass on because they are without any modifications.

The variable declarations for file reference variables in COBOL are listed in Example 8-5.

*Example 8-5 XML file reference filterable in COBOL*

```
01  DOC-AS-CLOB-FILE   IS SQL TYPE IS XML AS CLOB-FILE.
01  DOC-AS-BLOB-FILE   IS SQL TYPE IS XML AS BLOB-FILE.
01  DOC-AS-DBCLOB-FILE IS SQL TYPE IS XML AS DBCLOB-FILE.
```

These declarations are translated by the precompiler, as shown in Example 8-6.

*Example 8-6 XML file reference variables after transformation by the DB2 precompiler*

```
01  DOC-AS-CLOB-FILE.
  49 DOC-AS-CLOB-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC.
  49 DOC-AS-CLOB-FILE-DATA-LENGTH PIC S9(9) COMP-5.
  49 DOC-AS-CLOB-FILE-FILE-OPTION PIC S9(9) COMP-5.
  49 DOC-AS-CLOB-FILE-NAME PIC X(255).
01  DOC-AS-BLOB-FILE.
  49 DOC-AS-BLOB-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC.
  49 DOC-AS-BLOB-FILE-DATA-LENGTH PIC S9(9) COMP-5.
  49 DOC-AS-BLOB-FILE-FILE-OPTION PIC S9(9) COMP-5.
  49 DOC-AS-BLOB-FILE-NAME PIC X(255).
01  DOC-AS-DBCLOB-FILE.
  49 DOC-AS-DBCLOB-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC.
  49 DOC-AS-DBCLOB-FILE-DATA-LENGTH PIC S9(9) COMP-5.
  49 DOC-AS-DBCLOB-FILE-FILE-OPTION PIC S9(9) COMP-5.
  49 DOC-AS-DBCLOB-FILE-NAME PIC X(255).
```

To work with a file reference variable, you must initialize the fields concerning the file name, that is the name of the file and the length of the name, and the file option. The file option is used to signal which type of operation you want to perform on the file. These options are supplied as constant declarations in COBOL.

The possible values are shown in Table 8-2 on page 163. The length of the file is provided by DB2 when writing to a file and does not have to be provided by the application.

*Table 8-2 Options for file reference variables*

| FILE-OPTION | Constant | Operation |
|---|---|---|
| SQL-FILE-READ | 2 | Input from application to database. A regular file can be opened, read, and closed |
| SQL-FILE-CREATE | 4 | Output from database to application. Creates a new file if one does not exist. If one does exist, an error is returned. |
| SQL-FILE-APPEND | 8 | Output from database to application. Appends data to an existing file. If one does not exist, a new file is created. |
| SQL-FILE-OVERWRITE | 16 | Output from database to application. Overwrites an existing file. If one does not exist, a new file is created. |

The initialization needed to read from a file named ‘CAMT.STMT.XML’ is shown in Example 8-7.

*Example 8-7 Initialization of a file reference variable*

```
MOVE ‘CAMT.STMT.XML’ TO DOC-AS-CLOB-FILE-NAME.
MOVE 13 TO DOC-AS-CLOB-FILE-NAME-LENGTH.
MOVE SQL-FILE-READ TO DOC-AS-CLOB-FILE-OPTION.
```

When the file reference variable is subsequently referred to, the specified file option is executed. Therefore, if the file is used, such as in an insert operation, the file is read into DB2 and inserted without materializing in application memory.

**Note:** Apply PTF for currently open APAR PM25980 when using binary file reference and locator variables. It solves issues for not UTF-8-encoded XML files.

## 8.2 The BankStmt application in COBOL

This section contains implementation steps of the BankStmt application in COBOL.

The purpose of the application is to demonstrate how to use COBOL with DB2 pureXML. We emphasize the steps and choices that are related to XML, disregarding irrelevant code and components such as presentation layer or user dialog. The application might therefore seem simple compared to a real-life application.

We use the same subset of the ISO 20022 standard, namely the BankToCustomerStatement message, which is used throughout the book.

The BankToCustomerStatement message is sent by the account servicer to an account owner or to a party authorized by the account owner to receive the message. It is used to inform the account owner, or authorized party, of the entries booked to the account, and to provide the owner with balance information about the account at a given time.

For the COBOL application, we use the database schema that focuses on the statement as a business object. This approach implies that the BankToCustomerStatement is saved whole as one XML document, rather than shredding the message into smaller documents before saving them.

The level at which to define a document might be difficult to decide, but a good guideline is that if the XML document matches a business object, it makes a good basis for the application because it contains the data that is typically used in each application component.

## 8.2.1 Setting up the environment

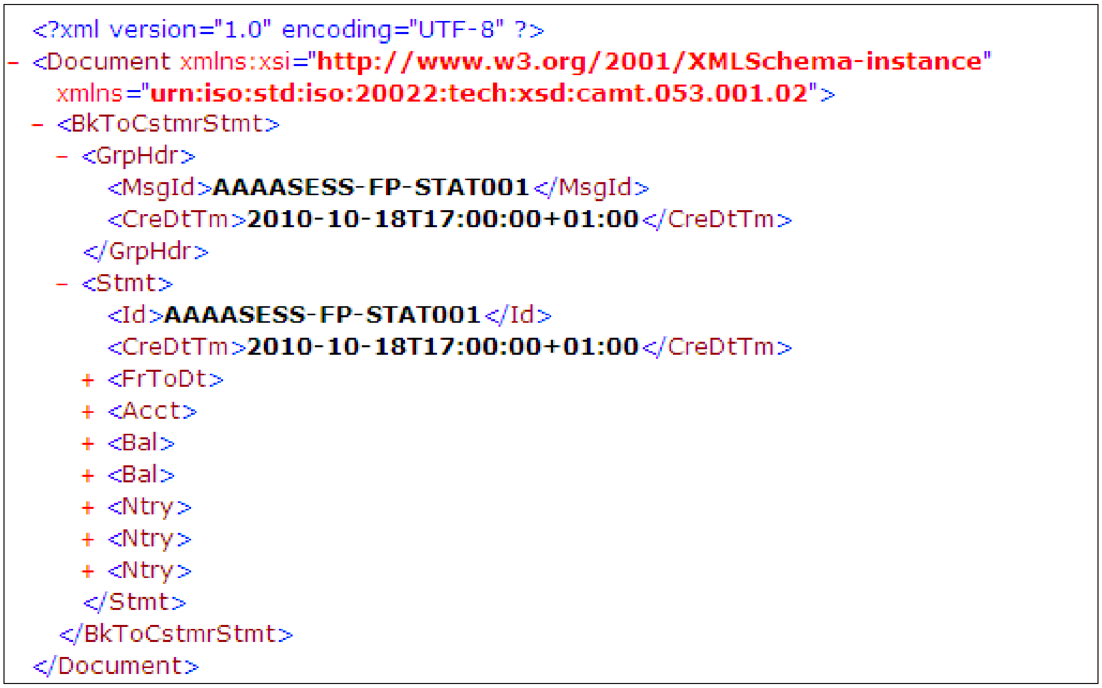
When creating a COBOL application, the initial work also includes setting up the environment. In this case, we need tables for holding the data, and the XML schema for validating the documents.

### The BankToCustomerStatement schema

To understand the structure of the documents, and to see the types of validation that the schema imposes on this structure, you start by looking at the schema for the BankToCustomerStatement.

Figure 8-2 shows the overall structure of a BankToCustomerStatement message as it looks in a browser. To expand the elements, click the plus sign (+); to collapse the elements, click the minus sign (-).

The outermost element is `Document` containing the `BkToCstmrStmt` element, which consists of a `GrpHdr` element and a `Stmt` element. The `Stmt` element contains simple `Id` elements and `CreDtTm`, complex `FrToDt` and `Acct` elements, and a number of complex `Bal` and `Ntry` elements.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <Document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02">
  - <BkToCstmrStmt>
    - <GrpHdr>
        <MsgId>AAAASESS-FP-STAT001</MsgId>
        <CreDtTm>2010-10-18T17:00:00+01:00</CreDtTm>
      </GrpHdr>
    - <Stmt>
        <Id>AAAASESS-FP-STAT001</Id>
        <CreDtTm>2010-10-18T17:00:00+01:00</CreDtTm>
      + <FrToDt>
      + <Acct>
      + <Bal>
      + <Bal>
      + <Ntry>
      + <Ntry>
      + <Ntry>
      </Stmt>
    </BkToCstmrStmt>
  </Document>
```

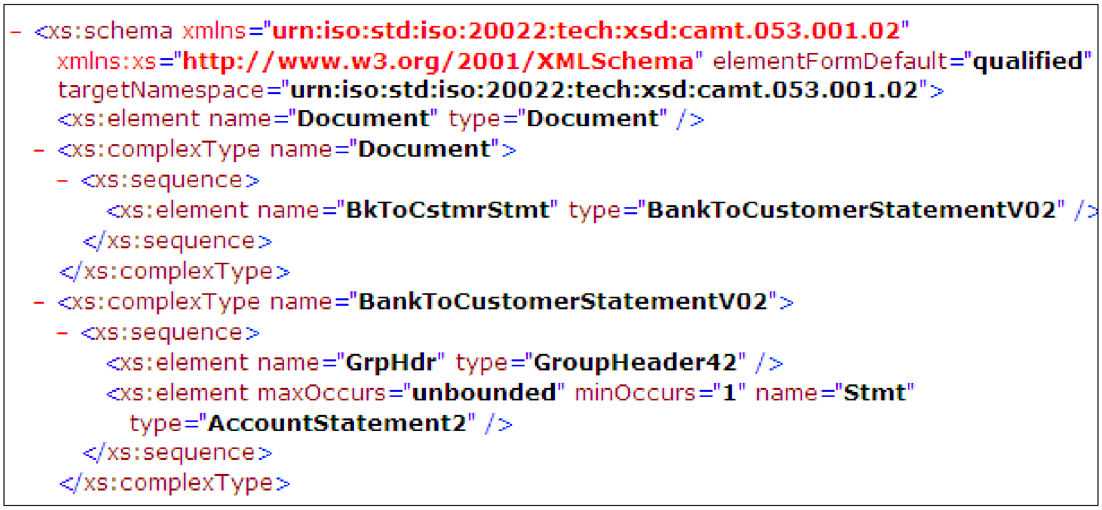
*Figure 8-2 BankToCustomerStatement message as shown in a browser*

The schema for messages of this format is available from the ISO 20022 website. It is extensive, and we do provide details, but we do examine a small subset of the schema that we work with in this book. If you are not used to working with XML schemas, it is still possible to get an idea of what the schema is saying. The schema itself is also written in XML.

Figure 8-3 shows the part of the schema that is concerned with the overall structure of the BankToCustomerStatement. The target namespace of the schema is as follows, and the outermost element is named Document with type Document:

```
urn:iso:std:iso:20022:tech:xsd:camt.053.001.02
```

The Document type is a complex type consisting of a sequence of elements with name `BkToCstmrStmt` of type `BankToCustomerStatementV02`, which is a complex type made up from another sequence of elements. This sequence contains an element of type `GrpHdr` and one or more occurrences of the `Stmt` element.



```
- <xs:schema xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    targetNamespace="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02">
    <xs:element name="Document" type="Document" />
  - <xs:complexType name="Document">
    - <xs:sequence>
        <xs:element name="BkToCstmrStmt" type="BankToCustomerStatementV02" />
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="BankToCustomerStatementV02">
    - <xs:sequence>
        <xs:element name="GrpHdr" type="GroupHeader42" />
        <xs:element maxOccurs="unbounded" minOccurs="1" name="Stmt"
          type="AccountStatement2" />
      </xs:sequence>
    </xs:complexType>
```

*Figure 8-3 Subset of the BankToCustomerStatement schema*

## Schema registration

We must register XML schemas in the DB2 XML schema repository to allow us to use DB2 to validate our XML documents.

The documents can either be validated explicitly by using the DSN_XMLVALIDATE function when inserting or updating, or DB2 can perform the validation automatically on insert and update. The latter requires that the XML column be associated with an XML type modifier that holds information about which schemas to use for validation.

We use automatic schema validation for the following reasons:

- The application code is simpler.
- It ensures that all documents are validated, although several applications might be inserting or updating the XML documents.
- If the schema changes, we only need to alter the XML type modifier on the table, not all the insert and update applications.

The schema consists of a single document (`camt.053.001.02.xsd`), from the ISO 20022 website, and is available on our workstation. Options to register the schema are as follows:

- Upload the schema to z/OS and register it through the command-line processor (CLP) in UNIX System Services.
- Register the schema through the CLP in Windows on our workstation, connecting to DB2 on z/OS by using DB2 Connect™.
- Use Optim Development Studio or similar development framework to register the schema.  This step also requires DB2 Connect to connect to DB2 for z/OS.

We choose to register the schema from the workstation using the CLP that is available in Windows. We first connect to DB2 using DB2 Connect, then register the schemas. The commands used are in Example 8-8.

For details about the other methods for schema registration, see 2.1.6, “XML schema repository and schema validation”

*Example 8-8 CLP commands for registration of XML schema*

```
db2 connect to DB0B user xmlr2 using passwd01
db2 register xmlschema 'camt.053.001.02.xsd' from 'camt.053.001.02.xsd' as
SYSXSR.camt_053_001_02
db2 complete xmlschema SYSXSR.camt_053_001_02
```

With the schema in place, we can create the tables needed for the application.

As stated earlier we have decided that the bank statement is what corresponds to a business object in this application, and we therefore need one table with one XML column, adding a few redundant columns for easy access and overview. The XML column is created with a type modifier that refers to the XML schema that we registered to allow for automatic schema validation.

The DDL for the tables is shown in Example 8-9. No indexes are created at this time. We add them when the programs and SQL queries are in place to make sure that the best possible indexes are chosen.

*Example 8-9 DDL for the table in the BankStmt application*

```
CREATE TABLE BK_TO_CSTMR_STMT
   (MSG_ID              VARCHAR(35) ,
    MSG_CRE_DT_TM       TIMESTAMP WITH TIME ZONE,
    BK_TO_CSTMR_STMT    XML
      (XMLSCHEMA ID SYSXSR.CAMT_053_001_02) NOT NULL) ;
```

## 8.2.2 Inserting XML documents

For the insert program, we assume that the full BankToCustomerStatement message is available in the file system. The insert program runs as a batch program, and the name of the file that is holding the XML document is passed as input in the JCL.

We also assume that the message is stored in EBCDIC, and matches the application code page so that the data is automatically converted correctly to UTF-8 if we use a single-byte character-based XML variable.

We need data from the document to populate the redundant columns in the base table. However, if we populate those after the insert task, we can do it in DB2 and we do not need to manipulate the contents of the XML file at all in the application program. This technique means that we can use a file reference variable to hold the XML.

The redundant columns are populated after the insert operation, so we only have to insert the XML document. We do, however, need to be able to identify the row after the insert to supply values for the redundant columns. Therefore, we use the following unique column, which is generated automatically by DB2 when inserting the XML document:

```
DB2_GENERATED_DOCID_FOR_XML COLUMN
```

We select this column from the final table of the insert statement.

The code needed for the initial insert is shown in Example 8-10.

*Example 8-10 Insert a BankToCustomerStatement*

```
01  WS-BLANK-STRING     PIC X(1) VALUE SPACES.
01  BK-STMT-DATA USAGE IS SQL TYPE IS XML AS CLOB-FILE.
01  DOC-ID              PIC S9(18) COMP-5.
...
ACCEPT BK-STMT-DATA-NAME FROM SYSIN.
INSPECT BK-STMT-DATA-NAME TALLYING BK-STMT-DATA-NAME-LENGTH
     FOR CHARACTERS BEFORE INITIAL WS-BLANK-STRING.
MOVE SQL-FILE-READ TO BK-STMT-DATA-FILE-OPTION.

EXEC SQL
  SELECT DB2_GENERATED_DOCID_FOR_XML INTO :DOC-ID
  FROM FINAL TABLE
  ( INSERT INTO BK_TO_CSTMR_STMT (BK_TO_CSTMR_STMT)
    VALUES (:BK-STMT-DATA) )
END-EXEC.
```

The XML data has now been validated and inserted, and all we need is to populate the two redundant columns. To do this step, we extract the contents from the XML document by using the XMLTABLE function.

The update, and therefore the remainder of what is needed for the insert operation is shown in Example 8-11.

*Example 8-11 Extracting key fields using XMLTABLE*

```
EXEC SQL
  UPDATE BK_TO_CSTMR_STMT ST1
  SET ( MSG_ID , MSG_CRE_DT_TM ) =
  ( SELECT X.MSGID , X.CREDTTM
  FROM BK_TO_CSTMR_STMT ST2,
   XMLTABLE (XMLNAMESPACES(DEFAULT
     'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02',
     '/Document/BkToCstmrStmt/GrpHdr'
   PASSING ST2.BK_TO_CSTMR_STMT
   COLUMNS
   "MSGID"     VARCHAR(35) PATH 'MsgId',
   "CREDTTM"   TIMESTAMP   PATH 'CreDtTm'
   ) X
   WHERE ST2.DB2_GENERATED_DOCID_FOR_XML = :DOC-ID )
   WHERE ST1.DB2_GENERATED_DOCID_FOR_XML = :DOC-ID
END-EXEC.
```

**Note:** XPath (and also XQuery) expressions are case-sensitive.

To test the program, two documents must be available in the file system: one that is valid according to the XML schema and one that is not.

The JCL to run the program is in Example 8-12. The name of the file that is holding the XML document is given as an input parameter in the SYSIN DD card.

*Example 8-12 JCL for running COBOL insert program*

```
//PH02CS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
```

```
//DBRMLIB  DD DISP=SHR,DSN=DB0BM.DBRMLIB.DATA
//STEPLIB  DD DISP=SHR,DSN=DB0BT.SDSNLOAD
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//REPORT   DD SYSOUT=*
//SYSIN    DD *
CAMT.BKSTMT.XML
//SYSTSIN DD *
DSN SYSTEM(DB0B)
 RUN  PROGRAM(INSBKST) PLAN(COBXML) -
      LIBRARY('DB0BM.RUNLIB.LOAD')
 END
//CARDIN   DD *
```

We first run it with the XML document that is not valid. To produce the non-valid document, we have omit the `GrpHdr` element altogether; the XML schema states that this element is required, as we saw in 8.2.1, “Setting up the environment” on page 164. Running the program with this document produces the error in Example 8-13.

*Example 8-13 Validation error on insert*

```
DSNT408I SQLCODE = -20399, ERROR:  ERROR ENCOUNTERED DURING XML VALIDATION: LOCATION 184;
TEXT An expected element match was not found.RC=0018,RSN=8604;
XSRID 144.
DSNT418I SQLSTATE   = 2201R SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNNOPAR SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = -510  0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'FFFFFE02'  X'00000000'  X'00000000'  X'FFFFFFFF'  X'00
         INFORMATION
```

The error is detected in location 184 of the document, and an expected element match was not found there. By inspecting the document, we find that this error is the first location after the start tag of the `BkToCstmrStmt`, and this is exactly the place where the `GrpHdr` should be, according to the schema.

We then run the program with the valid document, and the program completes as expected. To verify that the document has been validated, we use the DB2-supplied SQL function XMLXSROBJECTID, which takes an XML column and returns the XSR object identifier that was used to validate the XML document (or 0 if the document was not validated). The identifier can then be looked up in the SYSXSR.SYSOBJECTS catalog table as shown in Example 8-14.

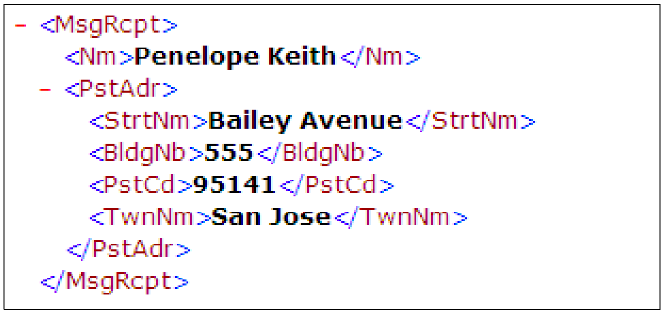
*Example 8-14 Determining whether an XML document has been validated*

```
---------+---------+---------+---------+---------+---------+-
SELECT B.ID, S.XSROBJECTNAME
FROM BK_TO_CSTMR_STMT B
,    SYSIBM.XSROBJECTS S
WHERE S.XSROBJECTID = XMLXSROBJECTID(B.BK_TO_CSTMR_STMT)
---------+---------+---------+---------+---------+---------+-
         ID  XSROBJECTNAME
---------+---------+---------+---------+---------+---------+-
          5  CAMT_053_001_02
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+-
```

### 8.2.3 Updating XML documents

We now examine a small program that updates a BankToCustomerStatement XML document.

A BankToCustomerStatement document can normally be sent either to the owner of the account or to another recipient. This option is modelled by having a MsgRcpt element in the GrpHdr of the document, which is filled in only if the recipient is not the owner. An example of a recipient is shown in Figure 8-4.



```
- <MsgRcpt>
    <Nm>Penelope Keith</Nm>
  - <PstAdr>
      <StrtNm>Bailey Avenue</StrtNm>
      <BldgNb>555</BldgNb>
      <PstCd>95141</PstCd>
      <TwnNm>San Jose</TwnNm>
    </PstAdr>
  </MsgRcpt>
```

*Figure 8-4 Message recipient of a BankToCustomerStatement*

To change the recipient of a BankToCustomerStatement, use the DB2 function XMLMODIFY, which can update part of an XML document.

The XMLMODIFY function has three subfunctions:

- Insert data into an XML document.
- Replace data in an XML document.
- Delete data from an XML document.

Depending on the following situations, we may use any one of the functions:

- To change the recipient from the owner to a non-owner, insert a MsgRcpt element.
- To change the recipient from a non-owner to another non-owner, replace the MsgRcpt element.
- To change the recipient from a non-owner to the owner, delete the MsgRcpt element.

> **Note:** The use of the XMLMODIFY function to update parts of an XML document is supported for tables with the multiversioning format that was introduced in DB2 10 only.
>
> A table has the multiversioning format in either of the following cases:
>
> - if it is created in DB2 10 NFM, has an XML column, and resides in a universal table space
> - If it is created in DB2 9, resides in a universal table space and all the XML columns are added to the table in DB2 10 NFM

For now, we assume that the MsgRcpt element is again available in a file that we can access through a file reference variable. We input to the program the name of this file, an ID of the XML message we want to change, and a choice of function to perform (replace, insert, or update).

An alternative to this approach is to input the raw data and then build an XML element by using COBOL features. See 8.3.1, “Generation of XML documents in COBOL” on page 178.

The updating program is shown in Example 8-15. The program inputs the MSG_ID of the row to update, a number indicating which function to perform, and the name of a file that is holding a MsgRcpt element. Depending on the function choice, the program executes one of three SQL statements, using XMLMODIFY to update the XML document:

- Inserts the MsgRcpt element.
- Replaces an existing MsgRcpt element.
- Deletes an existing MsgRcpt element.

*Example 8-15 COBOL program for updating a BkToCstmrStmt with a new MsgRcpt*

```
WORKING-STORAGE SECTION.
01  REPLACE-MSG-RCPT   PIC 9 VALUE 1.
01  INSERT-MSG-RCPT    PIC 9 VALUE 2.
01  DELETE-MSG-RCPT    PIC 9 VALUE 3.
01  FUNCTION-CHOICE    PIC 9.
      EXEC SQL INCLUDE SQLDA END-EXEC.
      EXEC SQL INCLUDE SQLCA END-EXEC.
01  WS-BLANK-STRING    PIC X(1) VALUE SPACES.
01  NEW-RCPT USAGE IS SQL TYPE IS XML AS CLOB-FILE.
01  MSGID              PIC X(35).
...
MAIN SECTION.
     EXEC SQL WHENEVER SQLERROR   GOTO DBERROR END-EXEC.
     EXEC SQL WHENEVER SQLWARNING GOTO DBERROR END-EXEC.

     ACCEPT MSGID FROM SYSIN.
     ACCEPT FUNCTION-CHOICE FROM SYSIN.
     ACCEPT NEW-RCPT-NAME FROM SYSIN.
     INSPECT NEW-RCPT-NAME TALLYING NEW-RCPT-NAME-LENGTH
          FOR CHARACTERS BEFORE INITIAL WS-BLANK-STRING.
     MOVE SQL-FILE-READ TO NEW-RCPT-FILE-OPTION.

     EVALUATE FUNCTION-CHOICE
         WHEN REPLACE-MSG-RCPT PERFORM REPLACE-RCPT
         WHEN INSERT-MSG-RCPT PERFORM INSERT-RCPT
         WHEN DELETE-MSG-RCPT PERFORM DELETE-RCPT
         WHEN OTHER DISPLAY "OTHER" FUNCTION-CHOICE
     END-EVALUATE.

REPLACE-RCPT.
      EXEC SQL
        UPDATE BK_TO_CSTMR_STMT
        SET BK_TO_CSTMR_STMT =
        XMLMODIFY (
        'declare default element namespace
-       '"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
-       'replace node
-       '/Document/BkToCstmrStmt/GrpHdr/MsgRcpt
-       'with $rcp/NewRcpt/MsgRcpt',
        :NEW-RCPT AS "rcp"
         )
        WHERE MSG_ID = :MSGID
      END-EXEC.

INSERT-RCPT.
       EXEC SQL
```

```
         UPDATE BK_TO_CSTMR_STMT
         SET BK_TO_CSTMR_STMT =
         XMLMODIFY (
         'declare default element namespace
-        '"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
-        'insert node $rcp/NewRcpt/MsgRcpt after
-        '/Document/BkToCstmrStmt/GrpHdr/CreDtTm',
         :NEW-RCPT AS "rcp"
          )
         WHERE MSG_ID = :MSGID
       END-EXEC.

DELETE-RCPT.
       EXEC SQL
         UPDATE BK_TO_CSTMR_STMT
         SET BK_TO_CSTMR_STMT =
         XMLMODIFY (
         'declare default element namespace
-        '"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
-        'delete node
-        '/Document/BkToCstmrStmt/GrpHdr/MsgRcpt'
          )
         WHERE MSG_ID = :MSGID
       END-EXEC.
```

Consider of the following information:

- For both the replace and insert expressions, a path of `$rcp/Newrcpt/MsgRcpt` is used instead of simply `$rcp` to indicate the new element. The reason is because we have wrapped the XML element in a container element, in this case NewRcpt, which is not to be inserted into DB2. This practice is necessary when inserting more than one element and therefore a good practice to adapt in general, for consistency.
- XPath expressions can be long, spanning several lines, especially if many namespace declarations are needed. The XPath expressions in the program are enclosed in apostrophes (single quotation marks), and an apostrophe is used together with the continuation character, which is a hyphen character (-), to indicate that the expression continues on the next line. This indication requires the program to be precompiled with the APOST and APOSTSQL options.

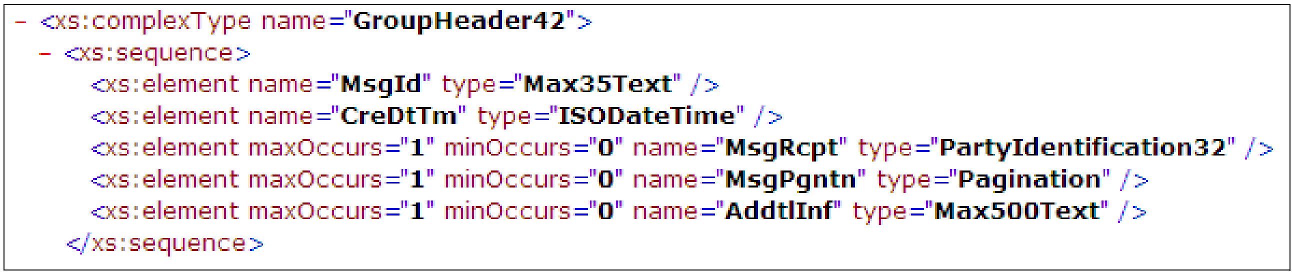
The program was tested with function 1, for inserting a new MsgRcpt element, and a file containing the MsgRcpt element, as shown in Figure 8-4 on page 169. This step produced the error in Example 8-16.

*Example 8-16 SQL error when updating XML document with MsgRcpt element*

```
DSNT408I SQLCODE = -20399, ERROR:  ERROR ENCOUNTERED DURING XML VALIDATION:
         LOCATION 237; TEXT An element is not in the choice.RC=0018,RSN=8608;
         XSRID 144.
DSNT418I SQLSTATE   = 2201R SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNNOPAR SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = -510 0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'FFFFFE02'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
```

The error indicates that an element, which begins in location 237 of the XML document, is not allowed in that place. This location turns out to be the MsgRcpt element, which is not surprising because this is the only place where we have changed the document.

The `MsgRcpt` element is inserted after the `CreDtTm` element in the `GrpHdr` element. We now compare this information to the schema definition of the `GrpHdr` element, which is shown in Figure 8-5. This definition is of the complex type `GroupHeader42`, which describes the `GrpHdr` element. We see that it consists of a sequence with a `MsgId` element, followed by a `CreDtTm` element, which again is followed by an optional `MsgRcpt` element and two other optional elements.



```
- <xs:complexType name="GroupHeader42">
  - <xs:sequence>
      <xs:element name="MsgId" type="Max35Text" />
      <xs:element name="CreDtTm" type="ISODateTime" />
      <xs:element maxOccurs="1" minOccurs="0" name="MsgRcpt" type="PartyIdentification32" />
      <xs:element maxOccurs="1" minOccurs="0" name="MsgPgntn" type="Pagination" />
      <xs:element maxOccurs="1" minOccurs="0" name="AddtlInf" type="Max500Text" />
    </xs:sequence>
```

*Figure 8-5 Schema definition for the GrpHdr element*

The location of the MsgRcpt element is not in conflict with the schema definition, so why does DB2 not recognize the element name as valid in the context?

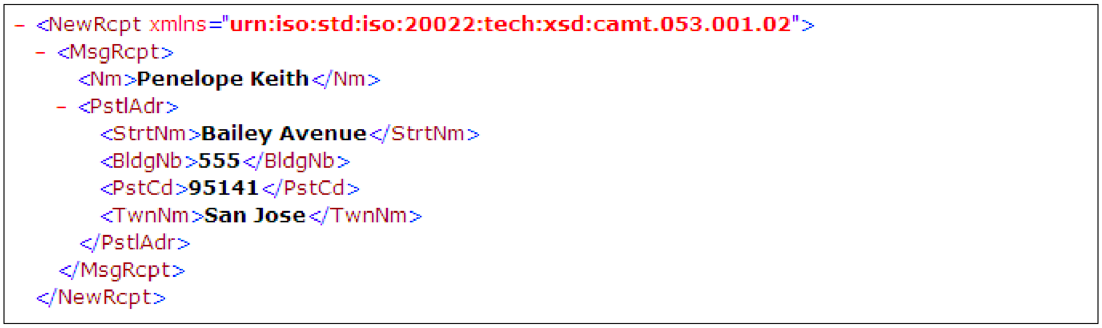
The explanation is related to namespaces. Recall that namespaces are a mechanism for ensuring uniqueness of element names, so that two elements in separate domains and possibly with separate structure and contents are not confused, although they have the same element name. By associating with each element a namespace, we guarantee that we know which one we are referencing. This association can be done either explicitly with a prefix to the element name, or implicitly by declaring a default element namespace.

The BankToCustomerStatement has the following default namespace and therefore all the elements belonging to this message have the same namespace unless another namespace is explicitly given:

```
urn:iso:std:iso:20022:tech:xsd:camt.053.001.02
```

The MsgRcpt element we tried to insert into the document had no namespaces associated with it, and therefore it is not the same element as the one in the BankToCustomerStatement schema. Therefore, it is not valid in the context.

We alter the contents of the file that contains the MsgRcpt element, shown in Figure 8-6, and attempt to run the update again. This time the namespace declaration matches the default element declaration, and the insert succeeds.



```
- <NewRcpt xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02">
  - <MsgRcpt>
      <Nm>Penelope Keith</Nm>
    - <PstlAdr>
        <StrtNm>Bailey Avenue</StrtNm>
        <BldgNb>555</BldgNb>
        <PstCd>95141</PstCd>
        <TwnNm>San Jose</TwnNm>
      </PstlAdr>
    </MsgRcpt>
  </NewRcpt>
```

*Figure 8-6 MsgRcpt element with namespace declaration*

### 8.2.4 Querying XML documents

Two options are available for retrieving XML from DB2:

- The data is retrieved as XML
- The data is converted to simple SQL types, either as a result of a cast operation, or from using the XMLTABLE function.

Example 8-17 shows how to select a whole XML document, with a given ID from the BK_TO_CSTMR_STMT table, and write it to a file by using a file reference variable.

*Example 8-17 Retrieval of an XML document to a file*

```
01  WS-BLANK-STRING    PIC X(1) VALUE SPACES.
01  MSGID              PIC X(35).
01  BK-STMT-FILE USAGE IS SQL TYPE IS XML AS CLOB-FILE.
      EXEC SQL INCLUDE SQLDA END-EXEC.
      EXEC SQL INCLUDE SQLCA END-EXEC.
...
MAIN SECTION.
     EXEC SQL WHENEVER SQLERROR   GOTO DBERROR END-EXEC.
     EXEC SQL WHENEVER SQLWARNING GOTO DBERROR END-EXEC.

     ACCEPT MSGID FROM SYSIN.

     ACCEPT BK-STMT-FILE-NAME FROM SYSIN.
     INSPECT BK-STMT-FILE-NAME TALLYING BK-STMT-FILE-NAME-LENGTH
          FOR CHARACTERS BEFORE INITIAL WS-BLANK-STRING.
     MOVE SQL-FILE-OVERWRITE TO BK-STMT-FILE-FILE-OPTION.

     EXEC SQL
       SELECT BK_TO_CSTMR_STMT INTO :BK-STMT-FILE
         FROM BK_TO_CSTMR_STMT
         WHERE MSG_ID = :MSGID
    END-EXEC.
```

We want to make a list of all entries that are created after a given date. The list is to contain the statement ID, amount, currency, credit-debit indicator, and datetime values.

We want to extract these values into simple SQL host variables of type DECIMAL, CHAR, and TIMESTAMP. In this case, it is transparent to the COBOL application that we are working with XML data, because all the XML manipulation takes place in DB2 through EXEC SQL statements. Even for the host variable declarations, we do not have to consider the various XML alternatives.

Example 8-18 on page 174 shows a program that extracts the entries for all bank statements and places them in relational host variables. The program inputs a time stamp and selects data from entries that have a time stamp later than this input time stamp. This step is done by passing the value of the time stamp as a parameter to the XPath expression, and then using it in the predicate where it is compared to the time stamp of each entry.

The example uses XMLTABLE to get a relational view of each entry, of which there may be several per BankToCustomerStatement, so potentially there are more rows returned than rows in the table. These rows are then filtered by the predicate, so potentially fewer rows might exist than rows in the table. These values are then placed in relational host variables for further processing.

*Example 8-18 Retrieval of data in relational format from an XML document*

```
01  FROM-TIME          PIC X(19).
01  STMT-ID            PIC X(20).
01  NTRY-TIME          PIC X(26).
01  AMOUNT             PIC S9(9)V9(2) COMP-3.
01  CREDIT-DEBIT       PIC X(4).
01  CURRENCY-NM        PIC X(4).
...
MAIN SECTION.
     EXEC SQL WHENEVER SQLERROR   GOTO DBERROR END-EXEC.
     EXEC SQL WHENEVER SQLWARNING GOTO DBERROR END-EXEC.

     ACCEPT FROM-TIME FROM SYSIN.

     EXEC SQL
       DECLARE C1 CURSOR FOR
        SELECT X.STMT, X.DTTM, X.CCY , X.AMT , X.CREDBTIND
        FROM BK_TO_CSTMR_STMT S,
        XMLTABLE (XMLNAMESPACES(DEFAULT
        'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02',
        '/Document/BkToCstmrStmt/Stmt/Ntry[BookgDt/DtTm>$tm]'
        PASSING S.BK_TO_CSTMR_STMT, TIMESTAMP(:FROM-TIME) AS "tm"
        COLUMNS
        "STMT"      CHAR(20)       PATH '../Id',
        "DTTM"      CHAR(26)       PATH 'BookgDt/DtTm',
        "CCY"       CHAR(4)        PATH 'Amt/@Ccy',
        "AMT"       DECIMAL(11,2)  PATH 'Amt',
        "CREDBTIND" CHAR(4)        PATH 'CdtDbtInd'
        ) X
     END-EXEC.
     EXEC SQL
       OPEN C1
     END-EXEC.
     EXEC SQL
       FETCH C1 INTO :STMT-ID ,
                     :NTRY-TIME ,
                     :CURRENCY-NM ,
                     :AMOUNT ,
                     :CREDIT-DEBIT
     END-EXEC.
     PERFORM WRITE-AND-FETCH
        UNTIL SQLCODE IS NOT EQUAL TO ZERO.
     EXEC SQL WHENEVER NOT FOUND  GOTO CLOSEC1    END-EXEC.
CLOSEC1.
     EXEC SQL CLOSE C1  END-EXEC.
```

### 8.2.5 Designing indexes

Recall that XML indexes can be used by XMLEXISTS and XMLTABLE functions, so only XPath patterns used in one of these functions are candidates for index use.

In the COBOL application, we have not used the XMLEXISTS function. The XMLTABLE function has been used twice, but only the one that extracts entries from the bank statement contains a predicate.

The pattern that is used for the context node in this expression is the candidate for index access, as Example 8-19 shows.

*Example 8-19 Candidate index pattern for the BankStmt application*

```
'/Document/BkToCstmrStmt/Stmt/Ntry[BookgDt/DtTm>$tm]'
```

We must include the path down to, and including, the DtTm element because this one is the one that the predicate evaluates, and we must include namespace declarations in the index. Example 8-20 shows the resulting index.

*Example 8-20 XML index on DtTm elements*

```
CREATE INDEX IXMLNTRY
ON BK_TO_CSTMR_STMT(BK_TO_CSTMR_STMT)
GENERATE KEY USING XMLPATTERN
'declare default element namespace
  "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
/Document/BkToCstmrStmt/Stmt/Ntry/BookgDt/DtTm'
AS SQL TIMESTAMP
```

**Note:** The support for date and time data types and functions in XML functions, including TIME, DATE, and TIMESTAMP as data types for indexes, requires DB2 10 NFM.

To ensure that the index is used by the COBOL program, we run the Runstats utility on the table and XML index, and then do an EXPLAIN of the program. Example 8-21 shows selection of several essential columns from the plan table for the program after and before the creation of the index.

The access type for access to BK_TO_CSTMR_STMTtable is DX, which is an indication that an XML index is being used for access; the access name is IXMLNTRY, which is the index we just created.

*Example 8-21 Access path using the index IXMLNTRY*

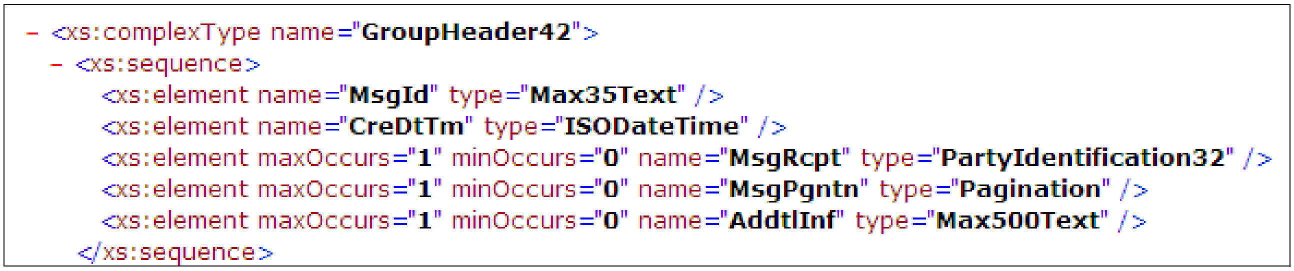
```
SELECT CREATOR, TNAME, METHOD, ACCESSTYPE, ACCESSCREATOR, ACCESSNAME
FROM PLAN_TABLE
WHERE PROGNAME = 'GETNTRY'
;
---------+---------+---------+---------+---------+---------+---------+---------+
CREATOR         TNAME          METHOD  ACCESSTYPE  ACCESSCREATOR   ACCESSNAME
---------+---------+---------+---------+---------+---------+---------+---------+
XMLR2           BK_TO_CSTMR_ST      0  DX          XMLR2           IXMLNTRY
XMLR2           X                   1  R
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
DSNE612I DATA FOR COLUMN HEADER TNAME COLUMN NUMBER 2 WAS TRUNCATED
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

## 8.2.6 Schema evolution

As part of the application life-cycle, an XML format is expected to evolve over time. This section has an example of a schema change, identifies which components are affected, and demonstrates how to change the application accordingly.

Recall that a BankToCustomerStatement can be sent either to the owner of the account or to another recipient by having a MsgRcpt element in the GrpHdr of the document, which is filled in only if the recipient is not the owner.

If you want to send the same message to more than one recipient, in the XML document you would have more than one MsgRcpt. What would be the effect on the schema and your applications? The schema definition of the GrpHdr is shown in Figure 8-7.

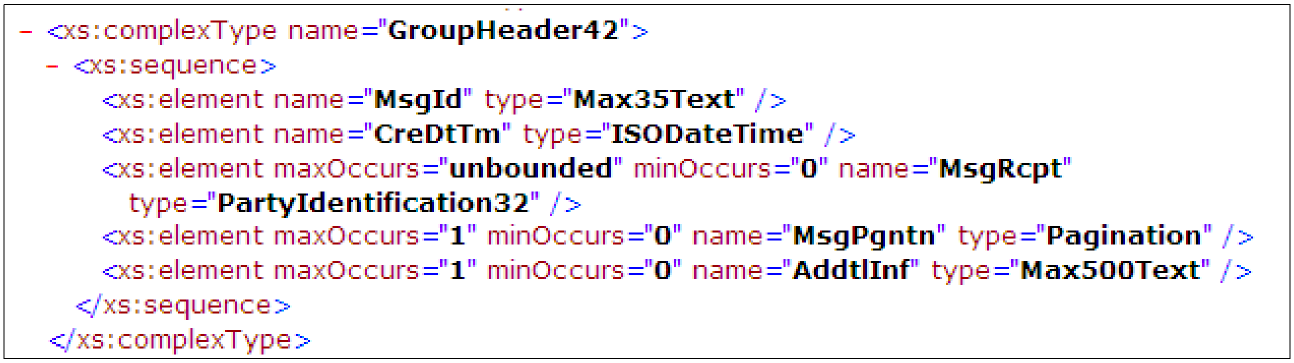


```
- <xs:complexType name="GroupHeader42">
  - <xs:sequence>
      <xs:element name="MsgId" type="Max35Text" />
      <xs:element name="CreDtTm" type="ISODateTime" />
      <xs:element maxOccurs="1" minOccurs="0" name="MsgRcpt" type="PartyIdentification32" />
      <xs:element maxOccurs="1" minOccurs="0" name="MsgPgntn" type="Pagination" />
      <xs:element maxOccurs="1" minOccurs="0" name="AddtlInf" type="Max500Text" />
    </xs:sequence>
```

*Figure 8-7 Schema definition of GrpHdr element*

Notice that the current schema definition does not allow more than one occurrence of the MsgRcpt element, which is determined by the `maxOccurs=`”1” attribute, and if you tried to insert a document containing, for example two message recipients, a validation error occurs. Therefore, the first item that has to change is the schema definition.

To alter the schema definition to allow for multiple occurrences of the MsgRcpt element, the only change necessary is to change the `maxOccurs=`”1” clause of the element to `maxOccurs=`”`unbounded`”. The result is shown in Figure 8-8.



```
- <xs:complexType name="GroupHeader42">
  - <xs:sequence>
      <xs:element name="MsgId" type="Max35Text" />
      <xs:element name="CreDtTm" type="ISODateTime" />
      <xs:element maxOccurs="unbounded" minOccurs="0" name="MsgRcpt"
        type="PartyIdentification32" />
      <xs:element maxOccurs="1" minOccurs="0" name="MsgPgntn" type="Pagination" />
      <xs:element maxOccurs="1" minOccurs="0" name="AddtlInf" type="Max500Text" />
    </xs:sequence>
  </xs:complexType>
```

*Figure 8-8 Revised schema definition for GrpHdr with multiple MsgRcpt elements*

We give the schema another version, for example `SYSXSR.CAMT_053_001_03` and register it to DB2 in the same way as we did in the original schema. See Example 8-8 on page 166 for details.

We then alter the table definition so that the version 3 schema is used for automatic validation. We do not want to remove the original schema because the data that is already in the table was validated against this schema, and removing it would remove the audit trail. It would also leave the table in a check-pending state because the rows would not have been validated against a schema mentioned in the XML type modifier. Instead, we add the new version to the type modifier of the XML column on top of the original schema.

When updating or inserting new XML documents, these documents are automatically validated against the latest version of the type modifier, when using automatic validation.

The DDL needed to extend the type modifier of the XML column is shown in Example 8-22.

*Example 8-22 Adding a new schema to an XML type modifier*

```
ALTER TABLE BK_TO_CSTMR_STMT
ALTER DOCUMENT
SET DATA TYPE XML
(XMLSCHEMA ID SYSXSR.CAMT_053_001_02,
ID SYSXSR.CAMT_053_001_03)
```

The table and schema repository are now set to handle the updated schema.

### Application changes

On the application side, the changes needed are of course influenced by the degree to which the fields involved in the change are used. The following examples nevertheless show that XML documents in many ways are robust to minor changes.

The insert application takes a whole XML document from the file system and inserts it into DB2 without manipulating it at all. The data validation is performed by DB2, but because we altered the XML type modifier to include the new schema, this step is already taken care of. Therefore, no changes are necessary in the insert application.

The update application alters the message recipient of the statement. Rather than altering it, we might, in the future want to only add another message recipient, because this is now allowed by the new schema. The COBOL program already contains the option of inserting a new message recipient; we investigate to determine whether the option can be used for inserting additional message recipients next to existing ones.

The XMLMODIFY expression that is used for inserting is shown in Example 8-23. It inserts the MsgRcpt element directly after the CreDtTm element. This approach means that if one or more MsgRcpt elements are already there, the new element is placed before these. This approach is valid according to the schema, so unless any significance is given to the order of the MsgRcpt elements, the application requires no changes to cater to the schema change.

Furthermore, because we wrap the MsgRcpt element in a container element, named NewRcpt, we are now also able to use the insert function to insert several new message recipients at one time, if we want.

*Example 8-23 Insert a MsgRcpt element after the CreDtTm element*

```
      EXEC SQL
         UPDATE BK_TO_CSTMR_STMT
         SET BK_TO_CSTMR_STMT =
         XMLMODIFY (
         'declare default element namespace
-        '"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
-        'insert node $rcp/NewRcpt/MsgRcpt after
-        '/Document/BkToCstmrStmt/GrpHdr/CreDtTm',
         :NEW-RCPT AS "rcp"
          )
         WHERE ID = :UPD-ID
       END-EXEC.
```

None of the query applications make use of the message recipients, so no changes are necessary here.

### Comparing to relational schema change

In summary, the only changes needed in the COBOL BankStmt application to allow multiple message recipients are the following changes in the environment:

- In the schema definition, change the attribute of the Msgrcpt element from `maxOccurs =“1”` to `maxOccurs =“unbounded”` setting.
- Register the resulting schema in DB2 with a new version number.
- Extend the type modifier of the XML column to include the new schema version.

These changes can all be performed as online changes. No changes are necessary on the application side.

If the database schema had been relational, the required changes would likely be much more extensive, involving a new table for the message recipient data and also converting existing data to make them available in the new table. This step, in turn, would require application changes, allowing the application to insert and update data in the new table.

# 8.3 COBOL functions for manipulating XML

COBOL also offers support for XML, independently of the pureXML support in DB2.

In certain cases, this support might complement the DB2 functionality, when basing the database design on pureXML.

This section briefly introduces the most important concepts, namely parsing, generating, and validating XML documents in native COBOL. For more details, see *Enterprise COBOL for z/OS Version 4.2 Programming Guide*, SC23-8529-01.

## 8.3.1 Generation of XML documents in COBOL

The DB2 pureXML functionality for creating XML documents from relational data is offered by various publishing functions such as XMLDOCUMENT, XMLELEMENT, XMLNAMESPACE, and others. These functions are described in 2.1.2, “SQL/XML language” on page 24.

COBOL also offers support for generation of XML documents from COBOL structures through the XML GENERATE statement.

This function takes as input a data item that is typically a group, and generates as output an XML document with similar structure as the input data item. The resulting element names are taken from the names in the group data item, and the resulting element contents is taken from the contents of these variables.

Example 8-24 on page 179 shows how to use XML GENERATE to generate a MsgRcpt element from a variable structure.

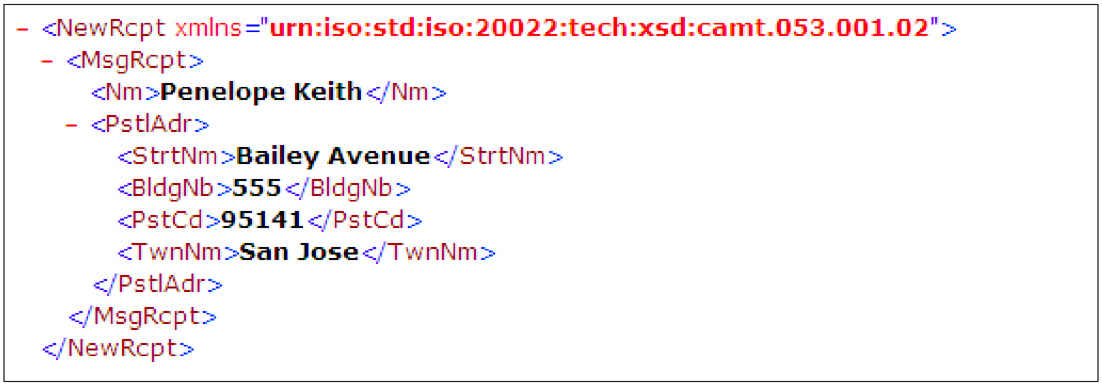
*Example 8-24 COBOL program for generation of the MsgRcpt element*

```
01 NewRcpt.
   02 MsgRcpt.
      05 Nm          PIC X(20) Value 'Pamela Woods'.
      05 PstlAdr.
         10 StrtNm   PIC X(20) Value '555'.
         10 BldgNb   PIC X(20) Value 'Bailey Avenue'.
         10 PstCd    PIC X(20) Value '95141'.
         10 TwnNm    PIC X(20) Value 'San Jose'.
01 NS               PIC X(50)
      VALUE 'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02'.
01 NEW-RCPT USAGE IS SQL TYPE IS XML AS CLOB(1K).
     EXEC SQL DECLARE :NEW-RCPT VARIABLE CCSID 1208 END-EXEC.
...
XML GENERATE NEW-RCPT-DATA FROM NewRcpt
COUNT IN NEW-RCPT-LENGTH
WITH ENCODING 1208
NAMESPACE IS NS
END-XML.
```

Consider the following information about the example:

- The result of the XML GENERATE operation is stored in the NEW-RCPT variable, defined as CLOB AS XML. To initialize the length of this variable, the keyword COUNT IN is used.
- The XML document is generated with code page UTF-8 to avoid code-page conversion. This process is done by use of the WITH ENCODING keyword in the XMLPARSE statement. To pass this information to DB2, an explicit CCSID declaration of the `:NEW-RCPT` host variable is made.
- The namespace is provided in a separate alphanumeric variable and included in the generation with the NAMESPACE IS keyword. This keyword is optional for the XML GENERATE statement.
- The element names are copied exactly as declared in the COBOL program so if mixed case element names are needed, be sure that the variable declarations are in mixed case.

Figure 8-9 shows the result of the generation.

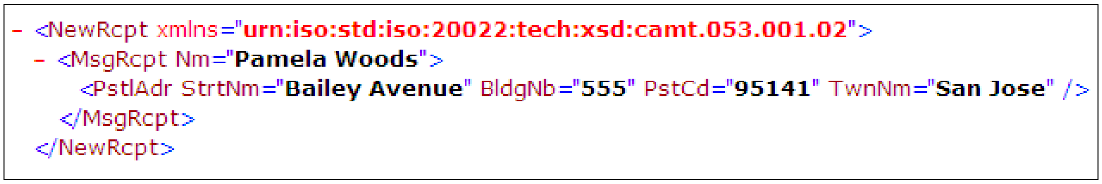


```
- <NewRcpt xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02">
  - <MsgRcpt>
      <Nm>Penelope Keith</Nm>
    - <PstlAdr>
        <StrtNm>Bailey Avenue</StrtNm>
        <BldgNb>555</BldgNb>
        <PstCd>95141</PstCd>
        <TwnNm>San Jose</TwnNm>
      </PstlAdr>
    </MsgRcpt>
  </NewRcpt>
```

*Figure 8-9 MsgRcpt element created with XML GENERATE*

The generation of the MsgRcpt element provides us with an alternative to the program for updating the MsgRcpt of a bank statement, described in 8.2.3, “Updating XML documents” on page 169. Rather than assuming that the new MsgRcpt is provided as an XML element in a text file, you may input the text values and build the XML element.

The XML GENERATE function also has the option of generating the values as attributes, instead of text elements, by using the keyword WITH ATTRIBUTES. This option results in the XML element shown in Figure 8-10.



```
- <NewRcpt xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02">
  - <MsgRcpt Nm="Pamela Woods">
      <PstlAdr StrtNm="Bailey Avenue" BldgNb="555" PstCd="95141" TwnNm="San Jose" />
    </MsgRcpt>
  </NewRcpt>
```

*Figure 8-10 MsgRcpt element created with XML GENERATE WITH ATTRIBUTES*

In most cases, the simple XML GENERATE is probably the better choice. Currently, there is no support for generating XML documents with both text elements and attributes.

### 8.3.2 Shredding XML documents in COBOL

DB2 offers support for shredding XML documents into relational data through the XMLTABLE function. This function assumes that the XML document is stored in DB2.

COBOL has similar support for XML documents that are stored in COBOL variables, and is offered through the XML PARSE statement.

This statement takes as input an XML document and a parsing procedure that handles the events that occur during parsing. With this approach, you can shred the document into COBOL variables, for example an alphanumeric group with the same structure as the XML document or separate elementary data items. In addition, you may process the data directly without saving the XML contents into variables.

The parsing procedure has to be written in the application program by using the various XML events provided by the XML parser as it goes through the document.

Example 8-25 shows how to use XML PARSE with a processing procedure to parse an MsgRcpt element into a variable structure.

*Example 8-25 COBOL program for shredding a MsgRcpt element into variables*

```
01 MsgRcpt.
   05 Nm PIC X(20).
   05 PstlAdr.
      10 StrtNm PIC X(20).
      10 BldgNb PIC X(20).
      10 PstCd PIC X(20).
      10 TwnNm PIC X(20).
01 NS                PIC X(50).
01 RCPT              PIC X(207).
01 CURRENT-ELEMENT   PIC X(40).
...
XML PARSE NEW-RCPT
  PROCESSING PROCEDURE GET-DATA
END-XML.
...
GET-DATA.
      EVALUATE XML-EVENT
           When 'START-OF-ELEMENT'
             Move XML-Text to current-element
           When 'CONTENT-CHARACTERS'
              EVALUATE current-element
                When 'Nm'
                  Move XML-TEXT TO Nm
                When 'PstlAdr'
                  Move XML-TEXT TO PstlAdr
                When 'StrtNm'
                  Move XML-TEXT TO StrtNm
                When 'BldgNb'
                  Move XML-TEXT TO BldgNb
                When 'PstCd'
                  Move XML-TEXT TO PstCd
                When 'TwnNm'
                  Move XML-TEXT TO TwnNm
                When Other
                  Continue
              End-evaluate
           When 'ATTRIBUTE-NAME'
              Continue
           When 'ATTRIBUTE-CHARACTERS'
              Continue
            When 'EXCEPTION'
              DISPLAY 'Exception code: ' XML-CODE
      END-EVALUATE.
```

The XML PARSE statement was introduced in COBOL before the pureXML format was available in DB2, and is useful for shredding XML documents that are saved as LOBs in DB2.

In general, save XML data as pureXML especially if you need to query the contents of that data, and in that case the shredding is more readily done by the XMLTABLE function in DB2.

## 8.3.3 Validation of XML documents in COBOL

Finally, COBOL also offers validation of XML documents against an XML schema through a variant of the XML PARSE statement. This approach requires Enterprise COBOL for z/OS V4.2.

The schema does not have to be registered anywhere, but it does have to be in a preprocessed format known as *Optimized Schema Representation (OSR)*. This preprocessing can be done from UNIX System Services with a command such as in Example 8-26. First, the schema is copied to UNIX System Services from TSO, and then the OSR document is generated.

*Example 8-26 Converting a schema to OSR format*

```
cp -B "//'XMLR2.BKSTMT.XSD'" /u/xmlr2/bkstmt.xsd
xsdosrg -v -o /u/xmlr2/bkstmt.osr /u/xmlr2/bkstmt.xsd
```

We extend the XML PARSE statement in Example 8-25 on page 181 with the validating phrase shown in Example 8-27.

*Example 8-27 XMLPARSE with schema validation*

```
CONFIGURATION SECTION.
SPECIAL-NAMES.
    XML-SCHEMA RSCHEMA IS 'DDSCHEMA'.
...
XML PARSE RCPT
  WITH ENCODING 1208
  VALIDATING WITH FILE RSCHEMA
  PROCESSING PROCEDURE GET-DATA
END-XML.
```

We use the schema declaration in the SPECIAL-NAMES section to associate the schema name RSCHEMA with an external file that contains the schema. This association can then be supplied as a DD statement in the JCL to run the COBOL program, as shown inExample 8-28.

*Example 8-28 DD statement for supplying a schema to the COBOL program*

```
//GO.DDSCHEMA DD PATH='/u/xmlr2/bkstmt.osr'
```

As mentioned previously, the automatic schema validation that can be set up in DB2, simply by associating an XML type identifier with the XML column, is robust to schema changes and can be easy to work with. This choice is generally better than explicit schema validation in both DB2 and COBOL.

# 9

# Utilities with XML

This chapter introduces the use of DB2 utilities with XML data types. Although the utilities handle XML objects in a way that is similar to how they handle LOB objects, for certain utilities you must specify certain XML keywords.

This chapter contains the following topics:

- CHECK DATA
- CHECK INDEX
- COPY
- COPYTOCOPY
- EXEC SQL
- LISTDEF
- LOAD
- MERGECOPY
- QUIESCE
- REBUILD INDEX
- RECOVER INDEX and RECOVER TABLESPACE
- REORG INDEX and REORG TABLESPACE
- REPAIR
- REPORT
- RUNSTATS
- UNLOAD
- DSNTIAUL
- DSN1COPY

We describe only those features that are directly related to XML.

© Copyright IBM Corp. 2011. All rights reserved.

## 9.1 CHECK DATA

Example 5-3 on page 78 and Example 5-4 on page 79 describes when an XML table space can be placed in a CHECK-pending state. Run the CHECK DATA utility to reset the CHECK-pending state.

The CHECK DATA utility checks XML relationships and can check the consistency between a base table space and the corresponding XML table spaces. If the base table space is not consistent with any related XML table spaces, CHECK DATA reports the error.

The default behavior of CHECK DATA is to check all objects that are in the CHECK-pending status (SCOPE PENDING). However, you may limit the scope of checking by specifying SCOPE REFONLY to check only the base tables, or SCOPE AUXONLY to check XML and LOB objects.

You may specify the action that DB2 performs when it finds an error in XML columns by using the XMLERROR keyword:

- XMLERROR REPORT reports only the error.
- XMLERROR INVALIDATE reports the error and sets the column in error to an invalid status.

You may specify the action that DB2 performs when it finds an error in LOB or XML columns by using the AUXERROR keyword:

- AUXERROR REPORT reports only the error.
- AUXERROR INVALIDATE reports the error and sets the column in error to an invalid status.

You do not normally specify both XMLERROR and AUXERROR.

The CHECK DATA utility has the following features to support XML data:

- Check consistency between the base table space and the NODEID index.
- Check consistency between the XML table space and the NODEID index.
- Check consistency in the document structure for each XML document.
- Validate schema if XML columns have a type modifier.

Figure 9-1 shows the CHECK DATA utility syntax for keywords that are introduced in DB2 10.

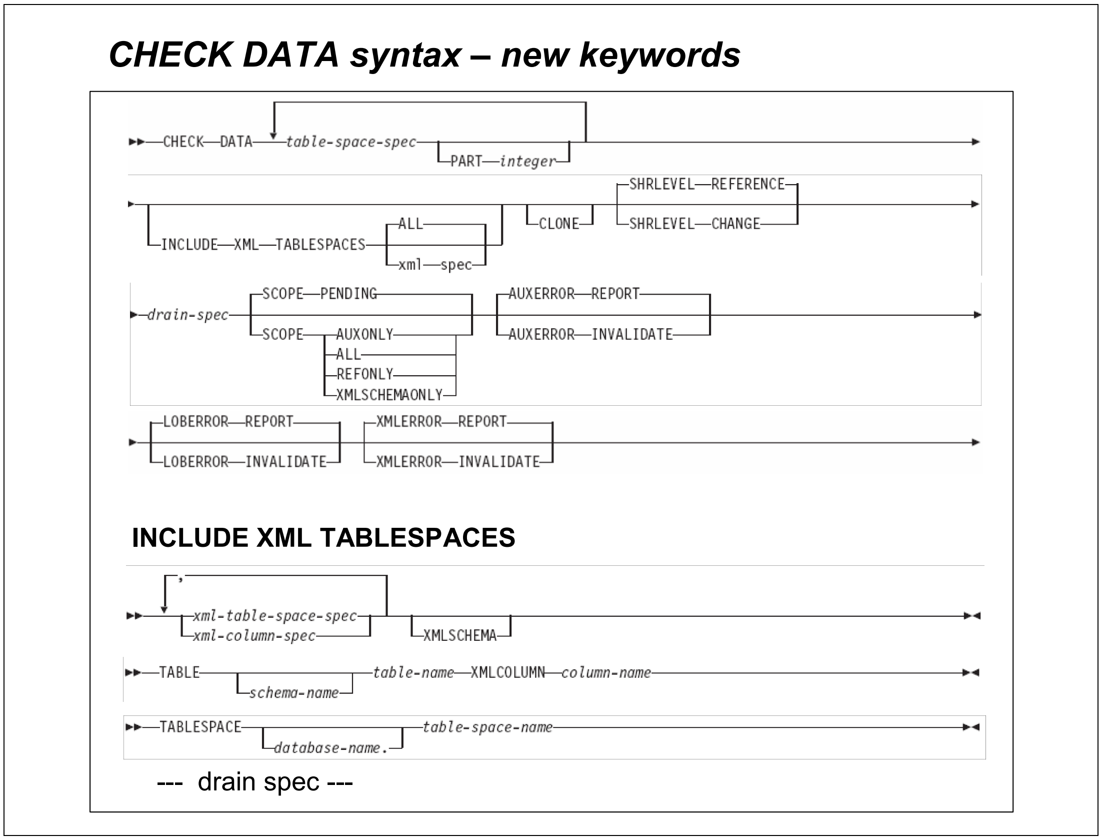

CHECK DATA syntax – new keywords

*Figure 9-1 CHECK DATA syntax: New keywords*

Options INCLUDE XML TABLESPACES and SCOPE XMLSCHEMAONLY are new.

If you specify the INCLUDE XML TABLESPACES option, CHECK DATA can check the structural integrity of XML documents. CHECK DATA can verify the following items for XML objects:

- All rows in an XML column exist in the XML table space.
- All documents in the XML table space are structurally valid.
- Each index entry in the NODEID index has a corresponding XML document.
- Each XML document in the XML table space has corresponding entries in the NODEID index.
- Each entry in the DOCID column in the base table space has a corresponding entry in the NODEID index over the XML table space, if the XML column is not null.
- Each entry in the NODEID index contains a corresponding value in the DOCID column.
- If an XML column has an XML type modifier, all XML documents in the column are valid with respect to at least one XML schema that is associated with the XML type modifier.

If the base table space is not consistent with any related XML table spaces, or a problem is found during any of the previously listed checks, CHECK DATA reports the error.

For XML checking, the default behavior of CHECK DATA is to check only the consistency between each XML column and its NODEID index. However, you may modify the scope of checking by specifying combinations of the CHECK DATA SCOPE keyword and the INCLUDE

XML TABLESPACES keyword. For LOBs, CHECK DATA utility checks for the consistency between the base table and the auxiliary index.

Table 9-1 is a reference table for the CHECK DATA invocation and is based on the combination of the various options that are applicable for LOBs also.

*Table 9-1 CHECK DATA invocation*

| CHECK DATA base table space | INCLUDE XML TABLESPACES | XMLSCHEMA | SCOPE | Structure check | Schema validation | XML checks | LOB checks |
|---|---|---|---|---|---|---|---|
| X | | | ALL/AUXONLY | - | - | Yes | Yes |
| X | | | REFONLY | - | - | - | - |
| X | | | XMLSCHEMA ONLY | - | Yes<br>Default: INCLUDE XML TABLESPACES ALL | - | - |
| X | | | PENDING | - | - | Yes<br>Base table spaces in CHKP/ACHKP | Yes<br>Base table spaces in CHKP/ACHKP |
| X | X | | ALL/AUXONLY | Yes<br>Specified XML table spaces only | - | Yes<br>All XML table spaces | Yes<br>All LOB table spaces |
| X | X | | REFONLY | - | - | - | - |
| X | X | | XMLSCHEMA ONLY | - | Yes<br>Specified XML table spaces only | - | - |
| X | X | | PENDING | - | Yes<br>Specified XML table spaces in CHKP | Yes<br>Base table spaces in CHKP/ACHKP | Yes<br>Base table spaces in CHKP/ACHKP |
| X | X | X | ALL/AUXONLY | Yes<br>Specified XML table spaces only | Yes<br>Specified XML table spaces only | Yes<br>All XML table spaces | Yes<br>All LOB table spaces |
| X | X | X | REFONLY | - | - | - | - |
| X | X | X | XMLSCHEMA ONLY | - | Yes<br>Specified XML table spaces only | - | - |
| X | X | X | PENDING | - | Yes<br>Specified XML table spaces in CHKP | Yes<br>Base table spaces in CHKP/ACHKP | Yes<br>Base table spaces in CHKP/ACHKP |

The XML checks column indicates CHECK DATA utility checks only for the consistency between the base table and the NODEID index.

The LOB checks column indicates CHECK DATA utility checks for the consistency between the base table and the auxiliary index.

Example 9-1 shows the CHECK DATA utility control statement for our scenario.

*Example 9-1 CHECK DATA utility control statement for scenario*

```
CHECK DATA TABLESPACE DSN00242.BKRTORCS SCOPE ALL
INCLUDE XML TABLESPACES(TABLE BK_TO_CSTMR_STMT XMLCOLUMN BK_TO_CSTMR_STMT) XMLSCHEMA
```

Table space DSN00242.BKRTORCS contains table BK_TO_CSTMR_STMT with XML column BK_TO_CSTMR_STMT, which has an XML type modifier. If you specify the statement shown in Example 9-1, CHECK DATA checks LOB relationships, the base table space, XML relationships, and the structural integrity of XML documents for column BK_TO_CSTMR_STMT, and does XML schema validation on the documents for column BK_TO_CSTMR_STMT.

You can use the SHRLEVEL REFERENCE or SHRELEVEL CHANGE option.

Considerations with the SHRLEVEL REFERENCE option are as follows:

- SHRLEVEL REFERENCE and AUXERROR/XMLERROR REPORT
  - If no problem is found, remove CHKP or ACHKP from table spaces
  - If problem is found:
    - DOCID of XML document is printed in the job output
    - No further action
- SHRLEVEL REFERENCE and AUXERROR/XMLERROR INVALIDATE
  - If no problem is found, remove CHKP or ACHKP from table spaces
  - If problem is found:
    - DOCID of XML document is printed in the job output
    - Exception tables automatically generated for XML column (schema validation only)
    - Affected XML documents moved to XML exception table (schema validation only)
    - Corrupted XML document deleted from XML table space
    - Index entries for corrupted XML documents removed from NODEID index
    - Invalid bit set in the XML indicator in the base table space
    - Value index(es) not touched/checked by CHECK DATA

Considerations (1 of 2) with option SHRLEVEL CHANGE are as follows:

- SHRLEVEL CHANGE in general
  - Utility creates shadow copies of all table and index spaces
  - Shadow copies discarded at the end of utility execution
- SHRLEVEL CHANGE and AUXERROR/XMLERROR REPORT
  - If no problem is found, CHKP or ACHKP remain on table spaces
  - If problem is found:
    - DOCID of XML document is printed in the job output
    - No further action

More considerations (2 of 2) with SHRLEVEL CHANGE are as follows:

- SHRLEVEL CHANGE and AUXERROR/XMLERROR INVALIDATE
  - If no problem is found, CHKP or ACHKP remain on table spaces
  - If problem is found:
    - DOCID of XML document is printed in the job output
    - For each corrupted XML document, CHECK DATA creates REPAIR LOCATE ... DOCID...DELETE
    - Message advises you to run REBUILD INDEX on NODEID index
    - NO RBDP set on NODEID index
    - REPAIR LOCATE … RID… REPLACE statements generated to invalidate entry
    - Value indexes not touched/checked by CHECK DATA
  - CHECK DATA generates REPAIR statements for XML documents that are not valid according to the defined XML type modifier also:
    - REPAIR LOCATE ... DOCID … DELETE
    - REPAIR LOCATE ... RID ... REPLACE
  - Run REPAIR
    - Delete corrupted XML documents from XML table space
    - REPAIR LOCATE TABLESPACE “DSN00155 "."XBKR0000"
    - DOCID 1 DELETE SHRLEVEL CHANGE
    - Set invalid bit in the XML indicator column in the base table space
    - REPAIR LOCATE TABLESPACE "DSN00155 "."BKRTORCS"
    - RID X'0123456789ABCDEF'
    - VERIFY OFFSET 28 DATA X'ABCD'
    - REPLACE OFFSET 28 DATA X'1234'

See 10.5, “Diagnostics” on page 242 for examples of invoking the CHECK DATA utility when diagnosing problems with XML data.

## 9.2 CHECK INDEX

Use the CHECK INDEX utility to check XML indexes, document ID indexes, and NODEID indexes. You do not have to specify any additional keywords. You cannot specify the DOCID and NODEID indexes in a single CHECK INDEX control statement because they belong to two separate table spaces. Specify the control statement by using two CHECK INDEX statements in the utility run as shown in Example 9-2.

*Example 9-2 CHECK INDEX utility JCL and output*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//DSNUPROC.SORTWK01 DD DSN=XMLR4.SORTWK01,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTWK02 DD DSN=XMLR4.SORTWK02,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTWK03 DD DSN=XMLR4.SORTWK03,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTWK04 DD DSN=XMLR4.SORTWK04,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSUT1 DD DSN=XMLR4.SYSUT1,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSIN    DD  *
CHECK INDEX (XMLR4.I_DOCIDBK_TO_CSTMR_STMT)
CHECK INDEX (XMLR4.I_NODEIDXBK_TO_CSTMR_STMT)

1DSNU000I    314 15:32:32.95 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   314 15:32:32.97 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    314 15:32:32.98 DSNUGUTC -  CHECK INDEX(XMLR4.I_DOCIDBK_TO_CSTMR_STMT)
 DSNU395I    314 15:32:32.99 DSNUKPIK - INDEXES WILL BE CHECKED IN PARALLEL, NUMBER OF
TASKS = 3
 DSNU701I  -DB0B 314 15:32:32.99 DSNUKIUL - 1 INDEX ENTRIES UNLOADED FROM
'DSN00242.BKRTORCS'
 DSNU705I    314 15:32:33.00 DSNUKPIK - UNLOAD PHASE COMPLETE - ELAPSED TIME=00:00:00
 DSNU719I    314 15:32:33.07 DSNUKPIK - 1 ENTRIES CHECKED FOR INDEX
'XMLR4.I_DOCIDBK_TO_CSTMR_STMT'
 DSNU720I    314 15:32:33.07 DSNUKPIK - SORTCHK PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU719I  -DB0B 314 15:32:33.07 DSNUKTER - 1 ENTRIES CHECKED FOR INDEX
'XMLR4.I_DOCIDBK_TO_CSTMR_STMT'
 DSNU380I  -DB0B 314 15:32:33.07 DSNUGSRX - TABLESPACE DSN00242.BKRTORCS PARTITION 1 IS IN
COPY PENDING

0DSNU050I    314 15:32:33.07 DSNUGUTC -  CHECK INDEX(XMLR4.I_NODEIDXBK_TO_CSTMR_STMT)
 DSNU395I    314 15:32:33.08 DSNUKPIK - INDEXES WILL BE CHECKED IN PARALLEL, NUMBER OF
TASKS = 3
 DSNU701I  -DB0B 314 15:32:33.08 DSNUKIUL - 1 INDEX ENTRIES UNLOADED FROM
'XMLR4.I_NODEIDXBK_TO_CSTMR_STMT'
```

```
 DSNU705I    314 15:32:33.09 DSNUKPIK - UNLOAD PHASE COMPLETE - ELAPSED TIME=00:00:00
 DSNU719I    314 15:32:33.12 DSNUKPIK - 1 ENTRIES CHECKED FOR INDEX
'XMLR4.I_NODEIDXBK_TO_CSTMR_STMT'
 DSNU720I    314 15:32:33.12 DSNUKPIK - SORTCHK PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU719I  -DBOB 314 15:32:33.12 DSNUKTER - 1 ENTRIES CHECKED FOR INDEX
'XMLR4.I_NODEIDXBK_TO_CSTMR_STMT'
 DSNU380I  -DBOB 314 15:32:33.12 DSNUGSRX - TABLESPACE DSN00242.XBKR0000 PARTITION 1 IS IN
COPY PENDING

NOTE: You can specify the CHECK INDEX control statement as shown below:
CHECK INDEX(ALL) TABLESPACE DSN00242.BKRTORCS
CHECK INDEX(ALL) TABLESPACE DSN00242.XBKR0000
The advantage with this approach is any user defined indexes are also checked.
```

After running the CHECK INDEX utility, you might need to correct XML data.

To correct XML data, based on the CHECK INDEX output, perform one of the actions listed in Table 9-2.

*Table 9-2 Action to be taken based on CHECK INDEX output*

| Problem | Action |
|---|---|
| Problem with a document ID index | 1. Confirm that the base table space is at the correct level.<br>2. Rebuild the index. |
| Problem with an XML table space for a NODEID index or an XML index and the index is correct | Run REPAIR LOCATE RID DELETE to remove the orphan row. |
| Problem with an XML table space for a NODEID index or an XML index and the index is incorrect | Run REBUILD INDEX or RECOVER INDEX to rebuild the index. |
| Problem with an XML index over an XML table space | Run REBUILD INDEX to rebuild the index.<br>Restriction: Do not run REPAIR LOCATE RID DELETE to remove orphan rows unless the NODEID index does not represent the same row and the base table space does not use the document ID index. |

> **Note:** CHECK INDEX of an XML index cannot run if REBUILD INDEX, REORG INDEX, or RECOVER is being run on that index because CHECK INDEX needs access to the NODEID index. CHECK INDEX SHRLEVEL CHANGE cannot run two jobs concurrently for two different indexes that are in the same table space.

## 9.3 COPY

Use the COPY utility to copy XML objects. You do not need to specify any additional keywords. When you specify that DB2 is to copy a table space with XML columns, DB2 does not automatically copy any related XML table spaces or indexes. You must explicitly specify the XML objects that you want to copy.

The COPY utility control statement in Example 9-3 specifies that DB2 is to copy base table space DSN00242.BKRTORCS and the XML table space DSN00242.XBKR0000.

*Example 9-3   COPY utility JCL for taking full image copy and output*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DBOBM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DBOB,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
TEMPLATE A DSN(&DB..&SN..&IC..D&DATE..T&TIME..COPY)
COPY TABLESPACE DSN00242.BKRTORCS COPYDDN(A)
     TABLESPACE DSN00242.XBKR0000 COPYDDN(A)

1DSNU000I    314 18:12:48.11 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   314 18:12:48.14 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    314 18:12:48.14 DSNUGUTC -  TEMPLATE A
DSN(&DB..&SN..&IC..D&DATE..T&TIME..COPY)
 DSNU1035I   314 18:12:48.14 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY

0DSNU050I    314 18:12:48.14 DSNUGUTC -  COPY TABLESPACE DSN00242.BKRTORCS COPYDDN(A)
TABLESPACE DSN00242.XBKR0000 COPYDDN(A)
 DSNU1038I   314 18:12:48.19 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=A
                        DDNAME=SYS00001
                        DSN=DSN00242.BKRTORCS.F.D2010314.T231248.COPY
 DSNU400I    314 18:12:48.24 DSNUBBID - COPY PROCESSED FOR TABLESPACE DSN00242.BKRTORCS
                       NUMBER OF PAGES=3
                       AVERAGE PERCENT FREE SPACE PER PAGE = 32.66
                       PERCENT OF CHANGED PAGES =  0.00
                       ELAPSED TIME=00:00:00
 DSNU1038I   314 18:12:48.27 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=A
                        DDNAME=SYS00002
                        DSN=DSN00242.XBKR0000.F.D2010314.T231248.COPY
 DSNU400I    314 18:12:48.30 DSNUBBID - COPY PROCESSED FOR TABLESPACE DSN00242.XBKR0000
                       NUMBER OF PAGES=3
                       AVERAGE PERCENT FREE SPACE PER PAGE = 19.66
                       PERCENT OF CHANGED PAGES =  0.00
                       ELAPSED TIME=00:00:00
 DSNU428I  -DBOB 314 18:12:48.31 DSNUBAFI - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE
DSN00242.BKRTORCS
 DSNU428I  -DBOB 314 18:12:48.31 DSNUBAFI - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE
DSN00242.XBKR0000
```

Both full and incremental image copies are supported for an XML table space, and also the SHRLEVEL REFERENCE, SHRLEVEL CHANGE, CONCURRENT, and FLASHCOPY options.

To demonstrate using COPY utility to take an incremental image copy, the XML document is modified, as shown in Figure 9-2 on page 192.

```
SELECT XMLSERIALIZE(
       XMLQUERY(
'declare default element namespace
"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
    /Document/BkToCstmrStmt/Stmt/Bal/Amt[../Tp/CdOrPrtry/Cd="CLBD"]'
  PASSING BK_TO_CSTMR_STMT) AS CLOB(500))
FROM BK_TO_CSTMR_STMT;

<Amt xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02"
Ccy="SEK">435678.50</Amt>
DSNE610I NUMBER OF ROWS DISPLAYED IS 1

UPDATE BK_TO_CSTMR_STMT
SET BK_TO_CSTMR_STMT = XMLMODIFY (
'declare default element namespace
"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
 replace value of node
/Document/BkToCstmrStmt/Stmt/Bal/Amt[../Tp/CdOrPrtry/Cd="CLBD"]
 with "900000"')
 WHERE MSG_ID IS NULL ;

DSNE615I NUMBER OF ROWS AFFECTED IS 1

SELECT XMLSERIALIZE(
       XMLQUERY(
'declare default element namespace
"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
    /Document/BkToCstmrStmt/Stmt/Bal/Amt[../Tp/CdOrPrtry/Cd="CLBD"]'
  PASSING BK_TO_CSTMR_STMT) AS CLOB(500))
FROM BK_TO_CSTMR_STMT;

<Amt xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02" Ccy="SEK">900000</Amt>
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
```

*Figure 9-2 Make a partial update to the XML document*

The COPY utility control statement in Example 9-4 on page 193 specifies that DB2 is to take an incremental image copy of base table space DSN00242.BKRTORCS and the XML table space DSN00242.XBKR0000.

*Example 9-4 COPY utility JCL for taking incremental image copy and output*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//DSNUPROC.SYSIN    DD  *
TEMPLATE A DSN(&DB..&SN..&IC..D&DATE..T&TIME..COPY)
COPY TABLESPACE DSN00242.BKRTORCS COPYDDN(A) FULL NO
     TABLESPACE DSN00242.XBKR0000 COPYDDN(A) FULL NO

1DSNU000I    314 18:14:25.22 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   314 18:14:25.25 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    314 18:14:25.26 DSNUGUTC -  TEMPLATE A
DSN(&DB..&SN..&IC..D&DATE..T&TIME..COPY)
 DSNU1035I   314 18:14:25.26 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY

0DSNU050I    314 18:14:25.26 DSNUGUTC -  COPY TABLESPACE DSN00242.BKRTORCS COPYDDN(A) FULL
NO TABLESPACE DSN00242.XBKR0000 COPYDDN(A) FULL NO
 DSNU1038I   314 18:14:25.30 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=A
                        DDNAME=SYS00001
                        DSN=DSN00242.BKRTORCS.I.D2010314.T231425.COPY
 DSNU400I    314 18:14:25.34 DSNUBBID - COPY PROCESSED FOR TABLESPACE DSN00242.BKRTORCS
                       NUMBER OF PAGES=3
                       AVERAGE PERCENT FREE SPACE PER PAGE = 32.66
                       PERCENT OF CHANGED PAGES =  5.88
                       ELAPSED TIME=00:00:00
 DSNU1038I   314 18:14:25.37 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=A
                        DDNAME=SYS00002
                        DSN=DSN00242.XBKR0000.I.D2010314.T231425.COPY
 DSNU400I    314 18:14:25.41 DSNUBBID - COPY PROCESSED FOR TABLESPACE DSN00242.XBKR0000
                       NUMBER OF PAGES=3
                       AVERAGE PERCENT FREE SPACE PER PAGE =  6.33
                       PERCENT OF CHANGED PAGES =  4.44
                       ELAPSED TIME=00:00:00
 DSNU428I  -DB0B 314 18:14:25.42 DSNUBAFI - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE
DSN00242.BKRTORCS
 DSNU428I  -DB0B 314 18:14:25.42 DSNUBAFI - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE
DSN00242.XBKR0000
```

Unless either the CONCURRENT option or the FLASHCOPY option is specified, COPY does not copy empty or unformatted data pages of an XML table space.

To copy an XML table space with a base table space that has the NOT LOGGED attribute, all associated XML table spaces must also have the NOT LOGGED attribute. The XML table space acquires this NOT LOGGED attribute by being linked to the logging attribute of its associated base table space. You cannot independently alter the logging attribute of an XML table space.

If the LOG column of the SYSIBM.SYSTABLESPACE record for an XML table space has the value of `X`, the logging attributes of the XML table space and its base table space are linked, and that the logging attribute of both table spaces is NOT LOGGED. To break the link, alter the logging attribute of the base table space back to LOGGED, and the logging attribute of both table spaces are changed back to LOGGED

## 9.4 COPYTOCOPY

Use the COPYTOCOPY utility to copy existing copies of the XML objects. You do not need to specify any additional keywords.

The COPYTOCOPY utility control statement in Example 9-5 specifies that DB2 is to make primary and backup copies for the remote site for the `DSN00242.XBKR0000` XML table space using the last full image copy that was created.

*Example 9-5 COPYTOCOPY utility JCL and output*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
TEMPLATE A DSN(&DB..&SN..&IC..&PB..D&DATE..T&TIME..COPY)
COPYTOCOPY TABLESPACE DSN00242.XBKR0000
FROMLASTFULLCOPY
RECOVERYDDN(A,A)

1DSNU000I    314 19:22:30.00 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   314 19:22:30.04 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    314 19:22:30.04 DSNUGUTC -  TEMPLATE A
DSN(&DB..&SN..&IC..&PB..D&DATE..T&TIME..COPY)
 DSNU1035I   314 19:22:30.04 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
1DSNU000I    314 19:22:31.81 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   314 19:22:31.83 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    314 19:22:31.83 DSNUGUTC -  COPYTOCOPY TABLESPACE DSN00242.XBKR0000
FROMLASTFULLCOPY RECOVERYDDN(A,A)
 DSNU1403I   314 19:22:31.87 DSNU2BCC - LOCAL SITE PRIMARY DATA SET
DSN00242.XBKR0000.F.D2010314.T233001.COPY WITH
                        START_RBA 0000696BCB4C IS IN USE BY COPYTOCOPY
                        FOR TABLESPACE DSN00242.XBKR0000
 DSNU1404I   314 19:22:31.88 DSNU2BDR - COPYTOCOPY PROCESSING COMPLETED FOR
                        TABLESPACE DSN00242.XBKR0000
                        ELAPSED TIME = 00:00:00
                        NUMBER OF PAGES COPIED=3
 DSNU1406I   314 19:22:31.89 DSNU2BDR - COPYTOCOPY COMPLETED. ELAPSED TIME = 00:00:00
```

## 9.5 EXEC SQL

The EXEC SQL utility control statement declares cursors or executes dynamic SQL statements. Use this utility as part of the DB2 cross-loader function of the LOAD utility.

With the cross-loader function, you can use a single LOAD job to transfer data from one location to another location or from one table to another table at the same location. You may use either a local server or any DRDA-compliant remote server as a data input source for populating your tables. Your input can even be from other sources besides DB2 for z/OS. You may use IBM Information Integrator Federation feature for access to data from sources as diverse as Oracle and Sybase, and also the entire DB2 family of database servers.

**Note:** You cannot declare a cursor that includes XML data. Thus, you cannot use the DB2 family cross-loader function to transfer data from XML columns. However, you can declare a cursor on a table with XML columns if the cursor does not include any XML columns.

For example, suppose that you create the following table with an XML column:

```
CREATE TABLE BK_TO_CSTMR_STMT
(MSG_ID VARCHAR(35),
 MSG_CRE_DT_TM TIMESTAMP WITH TIMEZONE,
 BK_TO_CSTMR_STMT XML NOT NULL)
```

You cannot declare the following cursor, because it includes XML data in the BK_TO_CSTMR_STMT column:

```
EXEC SQL
DECLARE C1 CURSOR FOR SELECT * FROM BK_TO_CSTMR_STMT
END-EXEC
```

However, you can declare a cursor that includes non-XML columns, as in the following example:

```
EXEC SQL
DECLARE C2 CURSOR FOR SELECT MSG_ID FROM BK_TO_CSTMR_STMT
END-EXEC
```

## 9.6 LISTDEF

When you create object lists with the LISTDEF utility, specify whether you want related XML objects to be included or excluded.

Use the following keywords to indicate the objects to include or exclude:

- ALL: Base and XML objects (This keyword is the default.)
- BASE: Base objects only
- XML: XML objects only

For example, the LISTDEF statements in Table 9-3 on page 196 generate the indicated lists.

*Table 9-3 Example LISTDEF statements*

| **LISTDEF statement followed by objects that are included in the list** |
|---|
| LISTDEF LISTALL INCLUDE TABLESPACES DATABASE DSN00242<br>INCLUDE INDEXSPACES DATABASE DSN00242<br>► All tables spaces in the DSN00242 database, including XML table spaces<br>► All index spaces in the DSN00242 database |
| LISTDEF LISTXML INCLUDE TABLESPACES DATABASE DSN00242 XML<br>INCLUDE INDEXSPACES DATABASE DSN00242 XML<br>► All XML table spaces in the DSN00242 database<br>► All XML index spaces in the DSN00242 database |
| LISTDEF LIST INCLUDE TABLESPACES DATABASE DSN00242 ALL<br>INCLUDE INDEXSPACES DATABASE DSN00242 ALL<br>EXCLUDE INDEXSPACES DATABASE DSN00242 XML<br>► All tables spaces in the DSN00242 database, including XML table spaces<br>► All index spaces in the DSN00242 database except for XML index spaces |

Example 9-6 shows the JCL for the LISTDEF utility for the first LISTDEF statement in Table 9-3 and the output of the utility run.

*Example 9-6 JCL for LISTDEF utility and output (1 of 3)*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
OPTIONS PREVIEW
LISTDEF LISTALL INCLUDE TABLESPACES DATABASE DSN00242
                INCLUDE INDEXSPACES DATABASE DSN00242

1DSNU000I    314 20:59:55.38 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   314 20:59:55.41 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    314 20:59:55.41 DSNUGUTC -  OPTIONS PREVIEW
 DSNU1000I   314 20:59:55.41 DSNUZODR - PROCESSING CONTROL STATEMENTS IN PREVIEW MODE
 DSNU1035I   314 20:59:55.41 DSNUZODR - OPTIONS STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    314 20:59:55.41 DSNUGUTC -  LISTDEF LISTALL INCLUDE TABLESPACES DATABASE
DSN00242 INCLUDE INDEXSPACES DATABASE DSN00242
 DSNU1035I   314 20:59:55.41 DSNUILDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
 DSNU1020I -DB0B 314 20:59:55.41 DSNUILSA - EXPANDING LISTDEF LISTALL
 DSNU1021I -DB0B 314 20:59:55.41 DSNUILSA -   PROCESSING INCLUDE CLAUSE DATABASE DSN00242.
 DSNU1022I -DB0B 314 20:59:55.42 DSNUILSA - CLAUSE IDENTIFIES 2 OBJECTS
 DSNU1021I -DB0B 314 20:59:55.42 DSNUILSA -   PROCESSING INCLUDE CLAUSE DATABASE DSN00242.
 DSNU1022I -DB0B 314 20:59:55.43 DSNUILSA - CLAUSE IDENTIFIES 2 OBJECTS
 DSNU1023I -DB0B 314 20:59:55.43 DSNUILSA - LISTDEF LISTALL CONTAINS 4 OBJECTS
 DSNU1010I   314 20:59:55.43 DSNUGPVV - LISTDEF LISTALL EXPANDS TO THE FOLLOWING OBJECTS:
             LISTDEF LISTALL -- 00000004 OBJECTS
               INCLUDE TABLESPACE DSN00242.BKRTORCS
               INCLUDE TABLESPACE DSN00242.XBKR0000
               INCLUDE INDEXSPACE DSN00242.IRDOCIDB
               INCLUDE INDEXSPACE DSN00242.IRNODEID
```

Example 9-7 shows the JCL for the LISTDEF utility for the second LISTDEF statement in Table 9-3 on page 196 and the output of the utility run.

*Example 9-7 JCL for LISTDEF utility and output (2 of 3)*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DBOBM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DBOB,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
OPTIONS PREVIEW
LISTDEF LISTXML INCLUDE TABLESPACES DATABASE DSN00242 XML
                INCLUDE INDEXSPACES DATABASE DSN00242 XML

1DSNU000I    314 21:00:53.19 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   314 21:00:53.22 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    314 21:00:53.22 DSNUGUTC -  OPTIONS PREVIEW
 DSNU1000I   314 21:00:53.22 DSNUZODR - PROCESSING CONTROL STATEMENTS IN PREVIEW MODE
 DSNU1035I   314 21:00:53.22 DSNUZODR - OPTIONS STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    314 21:00:53.22 DSNUGUTC -  LISTDEF LISTXML INCLUDE TABLESPACES DATABASE
DSN00242 XML INCLUDE INDEXSPACES DATABASE DSN00242 XML
 DSNU1035I   314 21:00:53.22 DSNUILDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
 DSNU1020I -DBOB 314 21:00:53.22 DSNUILSA - EXPANDING LISTDEF LISTXML
 DSNU1021I -DBOB 314 21:00:53.22 DSNUILSA -   PROCESSING INCLUDE CLAUSE DATABASE DSN00242.
 DSNU1022I -DBOB 314 21:00:53.22 DSNUILSA - CLAUSE IDENTIFIES 1 OBJECTS
 DSNU1021I -DBOB 314 21:00:53.22 DSNUILSA -   PROCESSING INCLUDE CLAUSE DATABASE DSN00242.
 DSNU1022I -DBOB 314 21:00:53.22 DSNUILSA - CLAUSE IDENTIFIES 1 OBJECTS
 DSNU1023I -DBOB 314 21:00:53.22 DSNUILSA - LISTDEF LISTXML CONTAINS 2 OBJECTS
 DSNU1010I   314 21:00:53.22 DSNUGPVV - LISTDEF LISTXML EXPANDS TO THE FOLLOWING OBJECTS:
             LISTDEF LISTXML -- 00000002 OBJECTS
               INCLUDE TABLESPACE DSN00242.XBKR0000
               INCLUDE INDEXSPACE DSN00242.IRNODEID
```

Example 9-8 shows the JCL for the LISTDEF utility for the third LISTDEF statement in Table 9-3 on page 196 and the output of the utility run.

*Example 9-8 JCL for LISTDEF utility and output (3 of 3)*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DBOBM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DBOB,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
OPTIONS PREVIEW
LISTDEF LIST INCLUDE TABLESPACES DATABASE DSN00242 ALL
             INCLUDE INDEXSPACES DATABASE DSN00242 ALL
             EXCLUDE INDEXSPACES DATABASE DSN00242 XML
1DSNU000I    314 21:02:09.82 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   314 21:02:09.85 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    314 21:02:09.85 DSNUGUTC -  OPTIONS PREVIEW
 DSNU1000I   314 21:02:09.85 DSNUZODR - PROCESSING CONTROL STATEMENTS IN PREVIEW MODE
 DSNU1035I   314 21:02:09.85 DSNUZODR - OPTIONS STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    314 21:02:09.85 DSNUGUTC -  LISTDEF LIST INCLUDE TABLESPACES DATABASE DSN00242
ALL INCLUDE INDEXSPACES
 DATABASE DSN00242 ALL EXCLUDE INDEXSPACES DATABASE DSN00242 XML
 DSNU1035I   314 21:02:09.85 DSNUILDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
 DSNU1020I -DBOB 314 21:02:09.85 DSNUILSA - EXPANDING LISTDEF LIST
 DSNU1021I -DBOB 314 21:02:09.85 DSNUILSA -   PROCESSING INCLUDE CLAUSE DATABASE DSN00242.
```

```
DSNU1022I -DB0B 314 21:02:09.86 DSNUILSA - CLAUSE IDENTIFIES 2 OBJECTS
DSNU1021I -DB0B 314 21:02:09.86 DSNUILSA -   PROCESSING INCLUDE CLAUSE DATABASE DSN00242.
DSNU1022I -DB0B 314 21:02:09.86 DSNUILSA - CLAUSE IDENTIFIES 2 OBJECTS
DSNU1021I -DB0B 314 21:02:09.86 DSNUILSA -   PROCESSING EXCLUDE CLAUSE DATABASE DSN00242.
DSNU1022I -DB0B 314 21:02:09.86 DSNUILSA - CLAUSE IDENTIFIES 1 OBJECTS
DSNU1023I -DB0B 314 21:02:09.86 DSNUILSA - LISTDEF LIST CONTAINS 3 OBJECTS
DSNU1010I   314 21:02:09.86 DSNUGPVV - LISTDEF LIST EXPANDS TO THE FOLLOWING OBJECTS:
            LISTDEF LIST -- 00000003 OBJECTS
              INCLUDE TABLESPACE DSN00242.BKRTORCS
              INCLUDE TABLESPACE DSN00242.XBKR0000
              INCLUDE INDEXSPACE DSN00242.IRDOCIDB
```

## 9.7 LOAD

Use one of the following methods to load data containing XML columns:

- The XML column can be loaded from the input record. The XML column value can be placed in the input record with or without any other loading column values. The input record can be in delimited or non-delimited format:
  - For a non-delimited format, the XML column is treated like a variable character with a 2-byte length preceding the XML value.
  - For a delimited format there are no length bytes present. If the input record is in spanned record format, specify the FORMAT SPANNED YES option.
- The XML column can be loaded from a separate file whether or not the XML column length is less than 32 KB.

To load data into a base table that has XML columns, use the following steps:

1. Create input data sets to ensure that you use the appropriate format:
   - If the data set is in delimited format, ensure that the XML input fields follow the standard LOAD utility delimited format.
   - If the data set is not in delimited format, specify the XML input fields similar to the way that you specify VARCHAR input. Specify the length of the field in a 2-byte binary field that precedes the data.
2. Create a LOAD utility control statement:
   - To load XML directly from input record, specify XML as the input field type. XML is the only acceptable field type and data type conversion is not supported. Do not specify DEFAULTIF.

     If you want the white space to be preserved in the XML data, also specify the keywords PRESERVE WHITESPACE. By default, LOAD strips the white space.

     When data in the binary XML format is loaded into a table, and PRESERVE WHITESPACE is not specified, DB2 strips white space only when the input data contains white space tags.
   - To load XML from a file, specify CHAR or VARCHAR along with either BLOBF, CLOBF or DBCLOBF to indicate that the input column contains a file name from which a BLOBF, CLOBF or DBCLOBF is to be loaded to the XML column.

3. Submit the utility control statement.

**Note:** When you load XML documents into a table, and the XML value cannot be cast to the type that you specified when you created the index, the value is ignored without any warnings or errors, and the document is inserted into the table.

When you insert XML documents into a table with XML indexes that are of type DECFLOAT, the values might be rounded when they are inserted. If the index is unique, the rounding might cause duplicates even if the original values are not exactly the same.

Example 9-9 shows the JCL for the LOAD utility and the output of the utility run.

*Example 9-9 LOAD utility JCL and output*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=&SYSUID,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//LOAD1 EXEC DSNUPROC,SYSTEM=DB0B,UID=''
//SYSREC   DD DSN=XMLR4.BKSTMT.XMLDATA,DISP=SHR
//SYSERR   DD DSN=XMLR4.LOAD.SYSERR,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//SYSDISC  DD DSN=XMLR4.LOAD.SYSDISC,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//SYSMAP   DD DSN=XMLR4.LOAD.SYSMAP,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//SYSUT1   DD DSN=XMLR4.LOAD.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//UTPRINT  DD SYSOUT=*
//SORTOUT  DD DSN=XMLR4.LOAD.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//DSNUPROC.SYSIN    DD  *
LOAD DATA  REPLACE
        INTO TABLE XMLR4.BK_TO_CSTMR_STMT
             (BK_TO_CSTMR_STMT POSITION(1) XML PRESERVE WHITESPACE)

1DSNU000I    300 20:39:12.45 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = XMLR4.XMLR4LD
 DSNU1044I   300 20:39:12.47 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    300 20:39:12.48 DSNUGUTC -  LOAD DATA REPLACE
 DSNU650I  -DB0B 300 20:39:12.48 DSNURWI -  INTO TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU650I  -DB0B 300 20:39:12.48 DSNURWI -   (BK_TO_CSTMR_STMT POSITION(1) XML PRESERVE
WHITESPACE)
 DSNU350I  -DB0B 300 20:39:13.13 DSNURRST - EXISTING RECORDS DELETED FROM TABLESPACE
 DSNU304I  -DB0B 300 20:39:13.26 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=1
FOR TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU1147I -DB0B 300 20:39:13.26 DSNURWT - (RE)LOAD PHASE STATISTICS - TOTAL NUMBER OF
RECORDS LOADED=1 FOR TABLESPACE DSN00242.BKRTORCS
 DSNU302I    300 20:39:13.27 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS
PROCESSED=1
 DSNU300I    300 20:39:13.27 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU349I  -DB0B 300 20:39:13.32 DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=1 FOR
INDEX XMLR4.I_DOCIDBK_TO_CSTMR_STMT
 DSNU258I    300 20:39:13.32 DSNURBXD - BUILD PHASE STATISTICS - NUMBER OF INDEXES=1
 DSNU259I    300 20:39:13.32 DSNURBXD - BUILD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU010I    300 20:39:13.33 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

```
NOTE: SYSREC shows the name of the data set which has the XML document. The first few
characters of the XML document are shown below:
Ù<?xml version="1.0" encoding="UTF-8" ?> <Document xmlns:xsi="http://ww ...
1F46A994A89A89977F4F748989889877EEC6F746644C98A989A4A999A7AA8778AA9766AA ...
1DCF74305592965EF1B0F055364957EF43608F0FE0C46344553074352A729EF8337A1166 ...
The first two positions contain X'11FD' which is the length of the XML document.
```

This technique requires specifying the exact size of the XML document preceding the data.

DB2 automatically generates the document ID column for each row that is loaded into a table with at least one XML column. The document ID column is partially hidden. It is not included in the result set of a `SELECT *` statement. However, you can query this column by name and view information about this column and its index in the catalog. Several utilities report information about this column in their output.

Loading XML data with the LOAD utility has the following restrictions:

- You cannot specify that XML input fields be loaded into non-XML columns, such as CHAR or VARCHAR columns.
- DB2 ignores any specified FREEPAGE and PCTFREE values until the next time that you run the REORG utility on this data.
- If you specify PREFORMAT, DB2 preformats the base table space, but not the XML table space.
- You cannot directly load the document ID column of the base table space.
- You cannot specify a default value for an XML column.
- You can load XML values that are greater than 32 KB by using file reference variables in the LOAD utility, or using applications with SQL XML as file reference variables.

## LOAD utility using file reference variable

The method of loading XML records using file reference variables is used when the XML records are stored in separate input files. The normal input file contains the data for the non-XML columns of the base table and the names of the XML files. The sum of the length of all normal data fields and the XML file names cannot exceed 32 KB.

The XML input files can be any of the following types:

- A sequential file
- A member of a PDS or PDSE
- A HFS file on a HFS directory
- A spanned file

The XML input file contains the entire XML record and the name of this file is stored in the normal load input file as a CHAR or VARCHAR field. Therefore, rather than containing the whole XML record, the normal input file now only contains a file name, which in most cases no longer causes the sequential file to reach the 32 KB limit.

Additional keywords have been added to the CHAR and VARCHAR field specifications of the LOAD utility to support a file name as the input for the actual XML record:

- BLOBF: The input field contains the name of a file with a BLOB value.
- CLOBF: The input field contains the name of a file with a CLOB value.
- DBCLOBF: The input field contains the name of a file with a DBCLOB value.

In case of CLOBF and DBCLOBF, CCSID conversions are done when the CCSID of the input data differs from the CCSID of the table space. (EBCDIC, ASCII, UNICODE, or CCSID keywords might have been specified for the input data; the default is EBCDIC input data.) In case of BLOBF, no conversions are done.

When the input field of a BLOBF, CLOBF, or DBCLOBF is NULL, the resulting XML value is NULL (null indicator field for the CHAR or VARCHAR field specified in the NULLIF keyword s hex FF).

Example 9-10 shows the JCL for the LOAD utility using file reference variable and the output of the utility run.

*Example 9-10 LOAD utility JCL (using file reference variable) and output*

```
//XMLR4LD  JOB (999,POK),'DB0B COBOL',CLASS=A,
// MSGCLASS=T,NOTIFY=&SYSUID,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//*
//LOAD1 EXEC DSNUPROC,SYSTEM=DB0B,UID=''
//SYSREC    DD DSN=XMLR4.CVXML.XMLTEXT,DISP=SHR
//SYSERR   DD DSN=XMLR4.LOAD.SYSERR,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//SYSDISC  DD DSN=XMLR4.LOAD.SYSDISC,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//SYSMAP   DD DSN=XMLR4.LOAD.SYSMAP,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//SYSUT1   DD DSN=XMLR4.LOAD.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//UTPRINT  DD SYSOUT=*
//SORTOUT  DD DSN=XMLR4.LOAD.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//DSNUPROC.SYSIN    DD  *
LOAD DATA  REPLACE
        INTO TABLE XMLR4.BK_TO_CSTMR_STMT
           (BK_TO_CSTMR_STMT POSITION(1:25) CHAR CLOBF)
/*

1DSNU000I    300 21:34:55.57 DSNUGTC - OUTPUT START FOR UTILITY, UTILID = XMLR4.XMLR4LD
 DSNU1044I   300 21:34:55.61 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    300 21:34:55.62 DSNUGUTC -  LOAD DATA REPLACE
 DSNU650I  -DB0B 300 21:34:55.62 DSNURWI -  INTO TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU650I  -DB0B 300 21:34:55.62 DSNURWI -   (BK_TO_CSTMR_STMT POSITION(1:25) CHAR CLOBF)
 DSNU350I  -DB0B 300 21:34:56.31 DSNURRST - EXISTING RECORDS DELETED FROM TABLESPACE
 DSNU304I  -DB0B 300 21:34:56.47 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=1
FOR TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU1147I -DB0B 300 21:34:56.47 DSNURWT - (RE)LOAD PHASE STATISTICS - TOTAL NUMBER OF
RECORDS LOADED=1 FOR TABLESPACE DSN00242.BKRTORCS
 DSNU302I    300 21:34:56.48 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS
PROCESSED=1
 DSNU300I    300 21:34:56.48 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU349I  -DB0B 300 21:34:56.54 DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=1 FOR
INDEX XMLR4.I_DOCIDBK_TO_CSTMR_STMT
 DSNU258I    300 21:34:56.54 DSNURBXD - BUILD PHASE STATISTICS - NUMBER OF INDEXES=1
 DSNU259I    300 21:34:56.54 DSNURBXD - BUILD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU010I    300 21:34:56.55 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

```
NOTE: SYSREC shows the name of the data set where positions 1 to 25 have the name of the
data set which has the XML document as shown below:
XMLR4.BKSTMT.XMLDATA1
The first few characters of the XML document are shown below:
<?xml version="1.0" encoding="UTF-8" ?> <Document xmlns:xsi="http://ww ...
46A994A89A89977F4F748989889877EEC6F746644C98A989A4A999A7AA8778AA9766AA ...
CF74305592965EF1B0F055364957EF43608F0FE0C46344553074352A729EF8337A1166 ...
The XML document starts from position 1.
```

## LOAD XML data using input in binary format

Example 9-11 shows the JCL for the LOAD utility and the output of the utility run.

*Example 9-11 LOAD utility JCL and output (input to load is in binary format)*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SORTWK01 DD DSN=XMLR4.SORTWK01,
//      DISP=(MOD,DELETE,CATLG),
//      SPACE=(16384,(20,20),,,ROUND),
//      UNIT=SYSDA
//DSNUPROC.SORTWK02 DD DSN=XMLR4.SORTWK02,
//      DISP=(MOD,DELETE,CATLG),
//      SPACE=(16384,(20,20),,,ROUND),
//      UNIT=SYSDA
//DSNUPROC.SORTWK03 DD DSN=XMLR4.SORTWK03,
//      DISP=(MOD,DELETE,CATLG),
//      SPACE=(16384,(20,20),,,ROUND),
//      UNIT=SYSDA
//DSNUPROC.SORTWK04 DD DSN=XMLR4.SORTWK04,
//      DISP=(MOD,DELETE,CATLG),
//      SPACE=(16384,(20,20),,,ROUND),
//      UNIT=SYSDA
//DSNUPROC.SYSREC DD DSN=XMLR4.UNLOAD.XML.BINARY.REC00,
//      DISP=OLD
//DSNUPROC.SYSUT1 DD DSN=XMLR4.SYSUT1,
//      DISP=(MOD,DELETE,CATLG),
//      SPACE=(16384,(20,20),,,ROUND),
//      UNIT=SYSDA
//DSNUPROC.SORTOUT DD DSN=XMLR4.SORTOUT,
//      DISP=(MOD,DELETE,CATLG),
//      SPACE=(16384,(20,20),,,ROUND),
//      UNIT=SYSDA
//DSNUPROC.SYSERR DD DSN=XMLR4.SYSERR,
//      DISP=(MOD,DELETE,CATLG),
//      SPACE=(16384,(20,20),,,ROUND),
//      UNIT=SYSDA
//DSNUPROC.SYSMAP DD DSN=XMLR4.SYSMAP,
//      DISP=(MOD,DELETE,CATLG),
//      SPACE=(16384,(20,20),,,ROUND),
//      UNIT=SYSDA
//DSNUPROC.SYSIN    DD  *
```

```
LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 EBCDIC  CCSID(00037,00000,00000)
 INTO TABLE "XMLR4"."BK_TO_CSTMR_STMT"
WHEN(00001:00002) = X'0003'
NUMRECS                    1
( "BK_TO_CSTMR_STMT"
 POSITION(  3)          XML PRESERVE WHITESPACE BINARYXML)
1DSNU000I    308 14:24:26.48 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   308 14:24:26.50 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    308 14:24:26.51 DSNUGUTC -  LOAD DATA INDDN SYSREC LOG NO RESUME YES
EBCDIC CCSID(37, 0, 0)
 DSNU650I  -DB0B 308 14:24:26.51 DSNURWI -  INTO TABLE "XMLR4". "BK_TO_CSTMR_STMT"
WHEN(1:2)=X'0003' NUMRECS 1
 DSNU650I  -DB0B 308 14:24:26.51 DSNURWI -   ("BK_TO_CSTMR_STMT" POSITION(3) XML
PRESERVE WHITESPACE BINARYXML)
 DSNU304I  -DB0B 308 14:24:26.70 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF
RECORDS=1 FOR TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU1147I -DB0B 308 14:24:26.70 DSNURWT - (RE)LOAD PHASE STATISTICS - TOTAL
NUMBER OF RECORDS LOADED=1 FOR TABLESPACE DSN00242.BKRTORCS
 DSNU302I    308 14:24:26.70 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF
INPUT RECORDS PROCESSED=1
 DSNU300I    308 14:24:26.70 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED
TIME=00:00:00
 DSNU349I  -DB0B 308 14:24:26.76 DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF
KEYS=1 FOR INDEX XMLR4.I_DOCIDBK_TO_CSTMR_STMT
 DSNU258I    308 14:24:26.76 DSNURBXD - BUILD PHASE STATISTICS - NUMBER OF
INDEXES=1
 DSNU259I    308 14:24:26.76 DSNURBXD - BUILD PHASE COMPLETE, ELAPSED
TIME=00:00:00
 DSNU380I  -DB0B 308 14:24:26.76 DSNUGSRX - TABLESPACE DSN00242.BKRTORCS PARTITION
1 IS IN COPY PENDING
 DSNU380I  -DB0B 308 14:24:26.76 DSNUGSRX - TABLESPACE DSN00242.XBKR0000 PARTITION
1 IS IN COPY PENDING
DSNU010I    308 14:24:26.77 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN
CODE=4
```

**Note:** The LOAD control statement is the output of the UNLOAD utility run, as shown in Example 9-26 on page 226. We set POSITION(3) because positions 3 and 4 contain the length of the XML document. Apply the PTF for APAR PM29986 if you want to use POSITION(*).

## LOAD XML data using input in spanned record format

Example 9-12 shows the JCL for the LOAD utility and the output of the utility run.

*Example 9-12 LOAD utility JCL and output (input to load is in spanned record format)*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SORTWK01 DD DSN=XMLR4.SORTWK01,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTWK02 DD DSN=XMLR4.SORTWK02,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTWK03 DD DSN=XMLR4.SORTWK03,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTWK04 DD DSN=XMLR4.SORTWK04,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSREC DD DSN=XMLR4.DSN00242.XBKR0000.T214620.UFILEREF.NEW,
//     DISP=OLD
//DSNUPROC.SYSUT1 DD DSN=XMLR4.SYSUT1,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTOUT DD DSN=XMLR4.SORTOUT,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSERR DD DSN=XMLR4.SYSERR,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSMAP DD DSN=XMLR4.SYSMAP,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSIN    DD  *
LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 EBCDIC  CCSID(00037,00000,00000)
 FORMAT SPANNED YES
 INTO TABLE "XMLR4"."BK_TO_CSTMR_STMT"
WHEN(00001:00002) = X'0003'
NUMRECS 1
( "DSN_NULL_IND_00001" POSITION(  00003)       CHAR(1)
, "MSG_ID"
 POSITION(  00004)       VARCHAR
                         NULLIF(DSN_NULL_IND_00001)=X'FF'
, "DSN_NULL_IND_00002" POSITION(  *)           CHAR(1)
, "MSG_CRE_DT_TM"
 POSITION(  *)           TIMESTAMP WITH TIME ZONE EXTERNAL(032)
                         NULLIF(DSN_NULL_IND_00002)=X'FF'
, "BK_TO_CSTMR_STMT"
```

```
 POSITION(  *)           XML PRESERVE WHITESPACE )

NOTE: The LOAD control statement is the output of UNLOAD utility run shown in Example 9-27 on page 227.
1DSNU000I    309 19:07:40.66 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   309 19:07:40.69 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    309 19:07:40.70 DSNUGUTC -  LOAD DATA INDDN SYSREC LOG NO RESUME YES EBCDIC CCSID(37, 0, 0) FORMAT SPANNED YES
 DSNU650I  -DB0B 309 19:07:40.70 DSNURWI -  INTO TABLE "XMLR4". "BK_TO_CSTMR_STMT" WHEN(1:2)=X'0003' NUMRECS 1
 DSNU650I  -DB0B 309 19:07:40.70 DSNURWI -   ("DSN_NULL_IND_00001" POSITION(3) CHAR(1),
 DSNU650I  -DB0B 309 19:07:40.70 DSNURWI -    "MSG_ID" POSITION(4) VARCHAR NULLIF(DSN_NULL_IND_00001)=X'FF',
 DSNU650I  -DB0B 309 19:07:40.70 DSNURWI -    "DSN_NULL_IND_00002" POSITION(*) CHAR(1),
 DSNU650I  -DB0B 309 19:07:40.70 DSNURWI -    "MSG_CRE_DT_TM" POSITION(*) TIMESTAMP WITH TIME ZONE EXTERNAL(32)
 NULLIF(DSN_NULL_IND_00002)=X'FF',
 DSNU650I  -DB0B 309 19:07:40.70 DSNURWI -    "BK_TO_CSTMR_STMT" POSITION(*) XML PRESERVE WHITESPACE)
 DSNU304I  -DB0B 309 19:07:40.89 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=1 FOR TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU1147I -DB0B 309 19:07:40.89 DSNURWT - (RE)LOAD PHASE STATISTICS - TOTAL NUMBER OF RECORDS LOADED=1 FOR TABLESPACE DSN00242.BKRTORCS
 DSNU302I    309 19:07:40.89 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=1
 DSNU300I    309 19:07:40.89 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU349I  -DB0B 309 19:07:40.95 DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=1 FOR INDEX XMLR4.I_DOCIDBK_TO_CSTMR_STMT
 DSNU258I    309 19:07:40.95 DSNURBXD - BUILD PHASE STATISTICS - NUMBER OF INDEXES=1
 DSNU259I    309 19:07:40.95 DSNURBXD - BUILD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU380I  -DB0B 309 19:07:40.96 DSNUGSRX - TABLESPACE DSN00242.BKRTORCS PARTITION 1 IS IN COPY PENDING
 DSNU380I  -DB0B 309 19:07:40.96 DSNUGSRX - TABLESPACE DSN00242.XBKR0000 PARTITION 1 IS IN COPY PENDING
DSNU010I    309 19:07:40.96 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4
```

For UNLOAD, SPANNED with BINARY XML performs significantly better than other formats. Use SPANNED if possible rather than file references.

## 9.8 MERGECOPY

The MERGECOPY utility performs either of the following merges:

- Merges incremental image copies to produce a merged incremental image copy,
- Merges full image copy with incremental image copies to produce a full image copy of the XML table spaces.

Example 9-13 shows the JCL for MERGECOPY utility and the output of the utility run. This utility run merges the full image copy taken in Example 9-3 on page 191 and the incremental image copy taken in Example 9-4 on page 193.

*Example 9-13 MERGECOPY utility JCL and output*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
TEMPLATE A DSN(&DB..&SN..&IC..D&DATE..T&TIME..COPY)
MERGECOPY TABLESPACE DSN00242.BKRTORCS COPYDDN(A) NEWCOPY YES
MERGECOPY TABLESPACE DSN00242.XBKR0000 COPYDDN(A) NEWCOPY YES

1DSNU000I    314 18:30:00.98 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   314 18:30:01.01 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    314 18:30:01.02 DSNUGUTC -  TEMPLATE A
DSN(&DB..&SN..&IC..D&DATE..T&TIME..COPY)
 DSNU1035I   314 18:30:01.02 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY

0DSNU050I    314 18:30:01.02 DSNUGUTC -  MERGECOPY TABLESPACE DSN00242.BKRTORCS COPYDDN(A)
NEWCOPY YES
 DSNU1038I   314 18:30:01.06 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=A
                        DDNAME=SYS00001
                        DSN=DSN00242.BKRTORCS.F.D2010314.T233001.COPY
 DSNU463I    314 18:30:01.07 DSNUYBR3 - THE PRIMARY IMAGE COPY DATA SET
DSN00242.BKRTORCS.I.D2010314.T231425.COPY WITH DATE=101110 AND TIME=181425
                       IS PARTICIPATING IN MERGECOPY.
 DSNU463I    314 18:30:01.09 DSNUYBR3 - THE PRIMARY IMAGE COPY DATA SET
DSN00242.BKRTORCS.F.D2010314.T231248.COPY WITH DATE=101110 AND TIME=181248
                       IS PARTICIPATING IN MERGECOPY.
 DSNU454I    314 18:30:01.13 DSNUYBRO - COPY MERGE COMPLETE
                       NUMBER OF COPIES=2
                       NUMBER OF COPIES MERGED=2
                       TOTAL NUMBER OF PAGES MERGED=6
                       ELAPSED TIME=00:00:00

0DSNU050I    314 18:30:01.14 DSNUGUTC -  MERGECOPY TABLESPACE DSN00242.XBKR0000 COPYDDN(A)
NEWCOPY YES
 DSNU1038I   314 18:30:01.17 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=A
                        DDNAME=SYS00004
                        DSN=DSN00242.XBKR0000.F.D2010314.T233001.COPY
 DSNU463I    314 18:30:01.18 DSNUYBR3 - THE PRIMARY IMAGE COPY DATA SET
DSN00242.XBKR0000.I.D2010314.T231425.COPY WITH DATE=101110 AND TIME=181425
                       IS PARTICIPATING IN MERGECOPY.
 DSNU463I    314 18:30:01.19 DSNUYBR3 - THE PRIMARY IMAGE COPY DATA SET
DSN00242.XBKR0000.F.D2010314.T231248.COPY WITH DATE=101110 AND TIME=181248
                       IS PARTICIPATING IN MERGECOPY.
 DSNU454I    314 18:30:01.23 DSNUYBRO - COPY MERGE COMPLETE
                       NUMBER OF COPIES=2
                       NUMBER OF COPIES MERGED=2
                       TOTAL NUMBER OF PAGES MERGED=6
                       ELAPSED TIME=00:00:00
```

## 9.9 QUIESCE

When you specify QUIESCE TABLESPACESET, the table space set includes related XML objects. You may specify that you want to quiesce the XML table space, base table space, or both of them, plus related index spaces if they are copy-enabled.

All table spaces that are involved in a versioning relationship are quiesced when QUIESCE is run on either the system-maintained temporal table or the history table space. Auxiliary LOB and XML table spaces on both system-maintained temporal table spaces and history table spaces are included.

Example 9-14 shows the JCL for the QUIESCE utility and the output of the utility run.

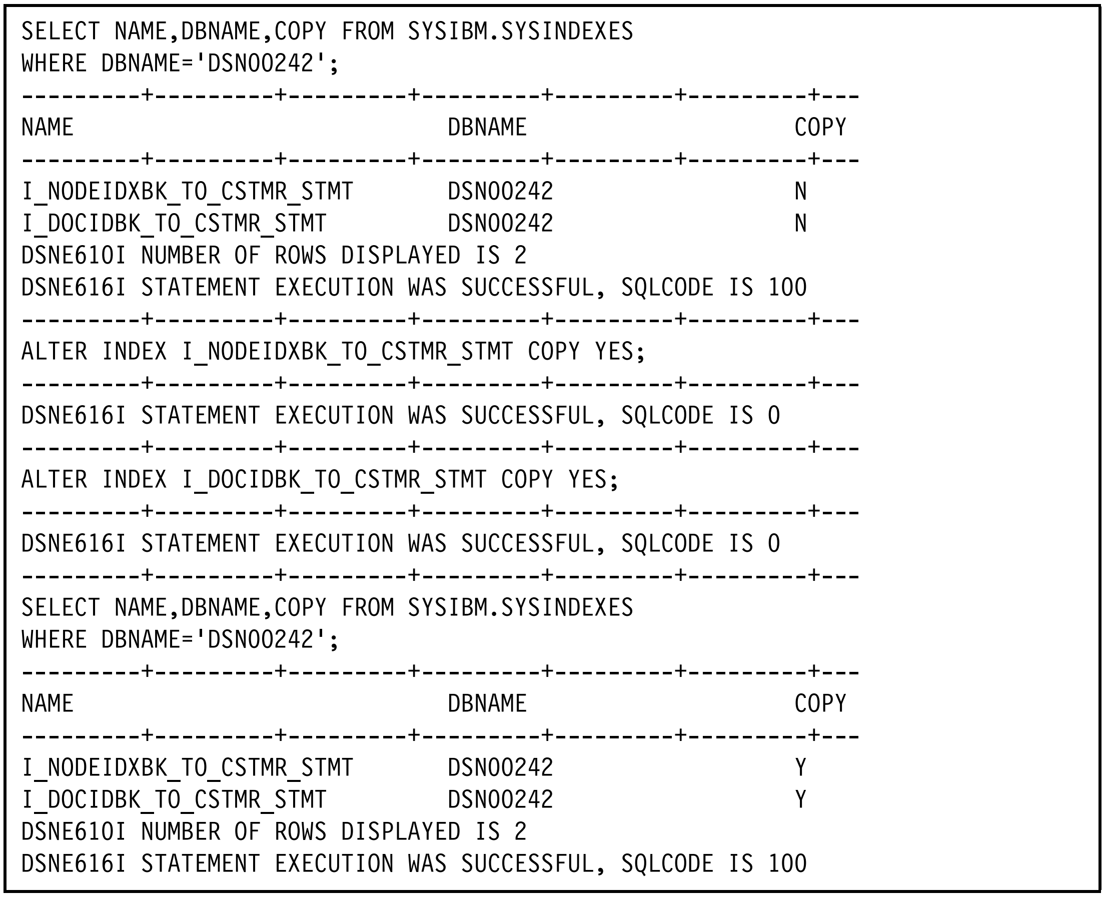
*Example 9-14 QUIESCE utility JCL and output (1 of 2)*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DBOBM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DBOB,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
QUIESCE TABLESPACESET DSN00242.BKRTORCS
1DSNU000I    314 21:50:34.75 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   314 21:50:34.77 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    314 21:50:34.78 DSNUGUTC -  QUIESCE TABLESPACESET DSN00242.BKRTORCS
 DSNU477I  -DBOB 314 21:50:34.78 DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACESET
DSN00242.BKRTORCS
```

```
 DSNU477I  -DBOB 314 21:50:34.78 DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE
DSN00242.BKRTORCS
 DSNU477I  -DBOB 314 21:50:34.78 DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE
DSN00242.XBKR0000
 DSNU474I  -DBOB 314 21:50:34.78 DSNUQUIA - QUIESCE AT RBA 000069A77034 AND AT LRSN
000069A77034
 DSNU475I    314 21:50:34.78 DSNUQUIB - QUIESCE UTILITY COMPLETE, ELAPSED TIME= 00:00:00
DSNU010I    314 21:50:34.79 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Notice that only the table spaces are included. Indexes are included only if they are copy-enabled.

Figure 9-3 shows how to alter the index definitions to make them copy-enabled.



```
SELECT NAME,DBNAME,COPY FROM SYSIBM.SYSINDEXES
WHERE DBNAME='DSN00242';
---------+---------+---------+---------+---------+---------+---
NAME                            DBNAME                  COPY
---------+---------+---------+---------+---------+---------+---
I_NODEIDXBK_TO_CSTMR_STMT       DSN00242                N
I_DOCIDBK_TO_CSTMR_STMT         DSN00242                N
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---
ALTER INDEX I_NODEIDXBK_TO_CSTMR_STMT COPY YES;
---------+---------+---------+---------+---------+---------+---
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---
ALTER INDEX I_DOCIDBK_TO_CSTMR_STMT COPY YES;
---------+---------+---------+---------+---------+---------+---
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---
SELECT NAME,DBNAME,COPY FROM SYSIBM.SYSINDEXES
WHERE DBNAME='DSN00242';
---------+---------+---------+---------+---------+---------+---
NAME                            DBNAME                  COPY
---------+---------+---------+---------+---------+---------+---
I_NODEIDXBK_TO_CSTMR_STMT       DSN00242                Y
I_DOCIDBK_TO_CSTMR_STMT         DSN00242                Y
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

*Figure 9-3   Copy enable the DOCID and NODEID indexes*

Example 9-15 shows the JCL for the QUIESCE utility and the output of the utility run.

*Example 9-15 QUIESCE utility JCL and output (2 of 2)*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
QUIESCE TABLESPACESET DSN00242.BKRTORCS
1DSNU000I    314 22:14:39.70 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   314 22:14:39.73 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    314 22:14:39.73 DSNUGUTC -  QUIESCE TABLESPACESET DSN00242.BKRTORCS
 DSNU477I  -DB0B 314 22:14:39.74 DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACESET
DSN00242.BKRTORCS
 DSNU477I  -DB0B 314 22:14:39.74 DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE
DSN00242.BKRTORCS
 DSNU477I  -DB0B 314 22:14:39.74 DSNUQUIA -    QUIESCE SUCCESSFUL FOR INDEXSPACE
DSN00242.IRDOCIDB
 DSNU477I  -DB0B 314 22:14:39.74 DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE
DSN00242.XBKR0000
 DSNU477I  -DB0B 314 22:14:39.74 DSNUQUIA -    QUIESCE SUCCESSFUL FOR INDEXSPACE
DSN00242.IRNODEID
 DSNU474I  -DB0B 314 22:14:39.74 DSNUQUIA - QUIESCE AT RBA 000069A99034 AND AT LRSN
000069A99034
 DSNU568I  -DB0B 314 22:14:39.74 DSNUGSRX - INDEX XMLR4.I_DOCIDBK_TO_CSTMR_STMT IS IN
INFORMATIONAL COPY PENDING STATE
 DSNU568I  -DB0B 314 22:14:39.74 DSNUGSRX - INDEX XMLR4.I_NODEIDXBK_TO_CSTMR_STMT IS IN
INFORMATIONAL COPY PENDING STATE
 DSNU475I    314 22:14:39.74 DSNUQUIB - QUIESCE UTILITY COMPLETE, ELAPSED TIME= 00:00:00
```

As expected, indexes are now included.

## 9.10 REBUILD INDEX

Use the REBUILD INDEX utility to rebuild XML indexes, DOCID indexes, and NODEID indexes. You do not need to specify any additional keywords in the REBUILD INDEX statement. SHRLEVEL CHANGE is not allowed on *not* logged tables and XML indexes.

When you process both NODEID indexes and XML indexes together, they are processed sequentially. First, the NODEID index is processed and then the XML index.

Example 9-16 shows the JCL for the REBUILD utility and the output of the utility run.

*Example 9-16 REBUILD INDEX utility JCL and output*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SORTWK01 DD DSN=XMLR4.SORTWK01,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTWK02 DD DSN=XMLR4.SORTWK02,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTWK03 DD DSN=XMLR4.SORTWK03,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTWK04 DD DSN=XMLR4.SORTWK04,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSUT1 DD DSN=XMLR4.SYSUT1,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSIN   DD  *
REBUILD INDEX (ALL) TABLESPACE DSN00242.BKRTORCS
REBUILD INDEX (ALL) TABLESPACE DSN00242.XBKR0000
1DSNU000I    315 19:54:04.76 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   315 19:54:04.79 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    315 19:54:04.79 DSNUGUTC -  REBUILD INDEX(ALL) TABLESPACE DSN00242.BKRTORCS
 DSNU3343I -DB0B 315 19:54:04.80 DSNUCINM - REAL-TIME STATISTICS INFORMATION MISSING FOR
TABLESPACE DSN00242.BKRTORCS PARTITION 1
 DSNU555I  -DB0B 315 19:54:04.81 DSNUCRUL - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
PROCESSED=1
 DSNU705I    315 19:54:04.81 DSNUCRIB - UNLOAD PHASE COMPLETE - ELAPSED TIME=00:00:00
 DSNU394I  -DB0B 315 19:54:05.01 DSNURBXC - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1 FOR
INDEX XMLR4.I_DOCIDBK_TO_CSTMR_STMT
 DSNU391I    315 19:54:05.01 DSNUCRIB - SORTBLD PHASE STATISTICS. NUMBER OF INDEXES = 1
 DSNU392I    315 19:54:05.01 DSNUCRIB - SORTBLD PHASE COMPLETE, ELAPSED TIME = 00:00:00
 DSNU425I  -DB0B 315 19:54:05.03 DSNUGFCR - INDEXSPACE DSN00242.IRDOCIDB   DOES NOT HAVE
THE COPY YES ATTRIBUTE
0DSNU050I    315 19:54:05.04 DSNUGUTC -  REBUILD INDEX(ALL) TABLESPACE DSN00242.XBKR0000
 DSNU3343I -DB0B 315 19:54:05.05 DSNUCINM - REAL-TIME STATISTICS INFORMATION MISSING FOR
TABLESPACE DSN00242.XBKR0000 PARTITION 1
 DSNU555I  -DB0B 315 19:54:05.06 DSNUCRUL - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
PROCESSED=1
 DSNU705I    315 19:54:05.06 DSNUCRIB - UNLOAD PHASE COMPLETE - ELAPSED TIME=00:00:00
 DSNU394I  -DB0B 315 19:54:05.27 DSNURBXC - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1 FOR
INDEX XMLR4.I_NODEIDXBK_TO_CSTMR_STMT
 DSNU391I    315 19:54:05.27 DSNUCRIB - SORTBLD PHASE STATISTICS. NUMBER OF INDEXES = 1
 DSNU392I    315 19:54:05.27 DSNUCRIB - SORTBLD PHASE COMPLETE, ELAPSED TIME = 00:00:00
 DSNU425I  -DB0B 315 19:54:05.28 DSNUGFCR - INDEXSPACE DSN00242.IRNODEID   DOES NOT HAVE
THE COPY YES ATTRIBUTE
DSNU010I    315 19:54:05.28 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

## 9.11 RECOVER INDEX and RECOVER TABLESPACE

Use the RECOVER utility to recover XML objects. You do not need to specify any additional keywords in the RECOVER statement. When you recover an XML table space or index to a point in time, be sure to recover all related objects to the same point in time. Related objects include XML objects, LOB objects, and referentially related objects. If you do not recover all related objects to the same point in time, one or more objects might be placed in a restrictive state.

For point-in-time recoveries of base, LOB, XML, and history objects, the VERIFYSET option, new with DB2 10, specifies whether the RECOVER utility verifies that all related objects that are required for the point-in-time recovery are included in the RECOVER control statement:

- VERIFYSET YES: The RECOVER utility verifies that all the objects that are required to perform a point-in-time recovery of the base, LOB, XML, and history objects, have been included in the RECOVER control statement. VERIFYSET YES is the default.
- VERIFYSET NO: The RECOVER utility does not verify that all the objects that are required to perform a point-in-time recovery of the base, LOB, XML, and history objects, have been included in the RECOVER control statement.

  By specifying VERIFYSET NO you can divide a point-in-time recovery into multiple jobs or avoid recovering objects that have changed since the selected recovery point.

The VERIFYSET option does not apply to point-in-time recoveries of catalog and directory objects.

The other option for point-in-time recovery is ENFORCE:

- ENFORCE YES: Specifies that CHKP and ACHKP pending states are set for a point-in-time recovery when only a subset of the related objects (BASE, LOB, XML, and RI) have been recovered to a point in time. ENFORCE YES is the default for catalog and directory objects. No override exists for the ENFORCE YES option for catalog and directory objects.
- ENFORCE NO: Specifies that CHKP and ACHKP pending states are not set for a point-in-time recovery when only a subset of the related objects (BASE, LOB, XML, and RI) have been recovered to a point in time.

We perform the following steps:

1. Take a full image copy of the base table space and XML table space (Example 9-3 on page 191).
2. Perform a partial update to the XML document (Example  on page 192).
3. Take an incremental image copy (Example 9-4 on page 193).
4. Merge the full and incremental image copies (Example 9-13 on page 205).

We now want to take the table space back to the full image copy before we perform the partial update to the XML document.

First however, we examine the status of the database and query the content of the table. Figure 9-4 shows the status of the database.

```
-DISPLAY DB(DSN00242)

DSNT360I  -DB0B ***********************************
DSNT361I  -DB0B *  DISPLAY DATABASE SUMMARY
                *    GLOBAL
DSNT360I  -DB0B ***********************************
DSNT362I  -DB0B     DATABASE = DSN00242  STATUS = RW
                   DBD LENGTH = 4028
DSNT397I  -DB0B
NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
BKRTORCS TS    0001 RW
BKRTORCS TS         RW
XBKR0000 XS    0001 RW
XBKR0000 XS         RW
IRDOCIDB IX   L0001 RW
IRDOCIDB IX    L*   RW
IRNODEID IX   L0001 RW
IRNODEID IX    L*   RW
******* DISPLAY OF DATABASE DSN00242 ENDED      **********************
DSN9022I  -DB0B DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
***
```

*Figure 9-4 Status of database DSN00242*

Figure 9-5 shows the content of the XML document.

```
SELECT XMLSERIALIZE(
       XMLQUERY(
'declare default element namespace
"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
    /Document/BkToCstmrStmt/Stmt/Bal/Amt[../Tp/CdOrPrtry/Cd="CLBD"]'
  PASSING BK_TO_CSTMR_STMT) AS CLOB(500))
FROM BK_TO_CSTMR_STMT;

<Amt xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02" Ccy="SEK">900000</Amt>
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
```

*Figure 9-5 Content of the XML document*

Example 9-17 on page 212 shows the JCL for RECOVER utility to recover the base table space to the LRSN value associated with the full image copy taken before the partial update to the XML document was done.

*Example 9-17   RECOVER TABLESPACE utility JCL and output*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
RECOVER TABLESPACE DSN00242.BKRTORCS TOLOGPOINT X'000069667C5E'
1DSNU000I    315 19:49:49.27 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   315 19:49:49.30 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    315 19:49:49.30 DSNUGUTC -  RECOVER TABLESPACE DSN00242.BKRTORCS TOLOGPOINT
X'000011112222'
 DSNU1316I -DB0B 315 19:49:49.31 DSNUCAIN - THE FOLLOWING TABLESPACES ARE MISSING FROM THE
RECOVERY LIST DSN00242.XBKR0000
 DSNU500I    315 19:49:49.31 DSNUCBDR - RECOVERY COMPLETE, ELAPSED TIME=00:00:00
DSNU012I    315 19:49:49.31 DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

**Note:** The RBA value for the TOLOGPOINT is the LRSN value associated with the full image copy shown in Example 9-21 on page 219. It is important to ensure that both the base table space and the XML table space are recovered to the same point-in-time.

Example 9-18 shows the JCL with the revised RECOVER utility control statement and output of the utility run.

*Example 9-18   RECOVER TABLESPACE utility JCL (and modified control statement) and output*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
RECOVER TABLESPACE DSN00242.BKRTORCS
        TABLESPACE DSN00242.XBKR0000
TOLOGPOINT X'000069667C5E'

1DSNU000I    315 19:36:29.07 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   315 19:36:29.10 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    315 19:36:29.10 DSNUGUTC -  RECOVER TABLESPACE DSN00242.BKRTORCS TABLESPACE
DSN00242.XBKR0000
 TOLOGPOINT X'000069667C5E'
 DSNU532I    315 19:36:29.11 DSNUCBMD - RECOVER TABLESPACE DSN00242.BKRTORCS   START
 DSNU515I    315 19:36:29.11 DSNUCBAL - THE IMAGE COPY DATA SET
DSN00242.BKRTORCS.F.D2010314.T231248.COPY WITH
 DATE=20101110 AND TIME=181248
                        IS PARTICIPATING IN RECOVERY OF TABLESPACE DSN00242.BKRTORCS
 DSNU504I    315 19:36:29.33 DSNUCBMD - MERGE STATISTICS FOR TABLESPACE DSN00242.BKRTORCS
-
                        NUMBER OF COPIES=1
                        NUMBER OF PAGES MERGED=3
                        ELAPSED TIME=00:00:00
 DSNU532I    315 19:36:29.34 DSNUCBMD - RECOVER TABLESPACE DSN00242.XBKR0000   START
 DSNU515I    315 19:36:29.34 DSNUCBAL - THE IMAGE COPY DATA SET
DSN00242.XBKR0000.F.D2010314.T231248.COPY WITH
 DATE=20101110 AND TIME=181248
                        IS PARTICIPATING IN RECOVERY OF TABLESPACE DSN00242.XBKR0000
 DSNU504I    315 19:36:29.54 DSNUCBMD - MERGE STATISTICS FOR TABLESPACE DSN00242.XBKR0000
-
```

```
                       NUMBER OF COPIES=1
                       NUMBER OF PAGES MERGED=3
                       ELAPSED TIME=00:00:00
 DSNU830I  -DBOB 315 19:36:29.14 DSNUCARS - INDEX XMLR4.I_DOCIDBK_TO_CSTMR_STMT IS IN
REBUILD PENDING
 DSNU831I  -DBOB 315 19:36:29.14 DSNUCARS - ALL INDEXES OF DSN00242.BKRTORCS ARE IN REBUILD
PENDING
 DSNU830I  -DBOB 315 19:36:29.36 DSNUCARS - INDEX XMLR4.I_NODEIDXBK_TO_CSTMR_STMT IS IN
REBUILD PENDING
 DSNU831I  -DBOB 315 19:36:29.36 DSNUCARS - ALL INDEXES OF DSN00242.XBKR0000 ARE IN REBUILD
PENDING
 DSNU1511I -DBOB 315 19:36:29.55 DSNUCALA - FAST LOG APPLY WAS NOT USED FOR RECOVERY
 DSNU1510I   315 19:36:29.55 DSNUCBLA - LOG APPLY PHASE COMPLETE, ELAPSED TIME = 00:00:00
 DSNU535I  -DBOB 315 19:36:29.55 DSNUCATM - FOLLOWING TABLESPACES RECOVERED TO A CONSISTENT
POINT
                        DSN00242.BKRTORCS
                        DSN00242.XBKR0000
 DSNU500I    315 19:36:29.57 DSNUCBDR - RECOVERY COMPLETE, ELAPSED TIME=00:00:00
```

The base table space and XML table space are now recovered to the same point-in-time. DB2 places the DOCID and NODEID indexes in REBUILD-pending state (RBDP) to ensure the indexes are consistent with the data.

Figure 9-6 shows the status of the database.

```
-DISPLAY DB(DSN00242)

DSNT360I  -DBOB ***********************************
DSNT361I  -DBOB *  DISPLAY DATABASE SUMMARY
                *    GLOBAL
DSNT360I  -DBOB ***********************************
DSNT362I  -DBOB     DATABASE = DSN00242  STATUS = RW
                   DBD LENGTH = 4028
DSNT397I  -DBOB
NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
BKRTORCS TS    0001 RW
BKRTORCS TS         RW
XBKR0000 XS    0001 RW
XBKR0000 XS         RW
IRDOCIDB IX   L0001 RW RBDP
IRDOCIDB IX    L*   RW RBDP
IRNODEID IX   L0001 RW RBDP
IRNODEID IX    L*   RW RBDP
******* DISPLAY OF DATABASE DSN00242 ENDED      **********************
DSN9022I  -DBOB DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
***
```

*Figure 9-6   Status of database DSN00242 (after partial recovery)*

We rebuild the indexes. Example 9-16 on page 208 shows the JCL for the REBUILD INDEX utility and the output of the utility run.

Figure 9-7 shows the status of the database after the indexes are rebuilt.

```
-DISPLAY DB(DSN00242)

DSNT360I  -DB0B ***********************************
DSNT361I  -DB0B *  DISPLAY DATABASE SUMMARY
                *    GLOBAL
DSNT360I  -DB0B ***********************************
DSNT362I  -DB0B     DATABASE = DSN00242  STATUS = RW
                   DBD LENGTH = 4028
DSNT397I  -DB0B
NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
BKRTORCS TS    0001 RW
BKRTORCS TS         RW
XBKR0000 XS    0001 RW
XBKR0000 XS         RW
IRDOCIDB IX   L0001 RW
IRDOCIDB IX    L*   RW
IRNODEID IX   L0001 RW
IRNODEID IX    L*   RW
******* DISPLAY OF DATABASE DSN00242 ENDED      **********************
DSN9022I  -DB0B DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
***
```

*Figure 9-7 Status of database DSN00242 (after partial recovery and rebuild of indexes)*

Figure 9-8 shows the content of the XML document.

```
SELECT XMLSERIALIZE(
       XMLQUERY(
'declare default element namespace
"urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
    /Document/BkToCstmrStmt/Stmt/Bal/Amt[../Tp/CdOrPrtry/Cd="CLBD"]'
  PASSING BK_TO_CSTMR_STMT) AS CLOB(500))
FROM BK_TO_CSTMR_STMT;

<Amt xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02"
Ccy="SEK">435678.50</Amt>
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
```

*Figure 9-8 Content of the XML document after partial recovery*

> **Note:** If the DOCID and NODEID indexes are enabled for image copy, Figure 9-6 on page 213 would show RECP instead of RBDP. Instead of rebuilding the indexes you would recover the indexes by using RECOVER INDEX utility, which uses the image copy you would have established for these indexes by using the COPY utility.

## 9.12 REORG INDEX and REORG TABLESPACE

Use the REORG INDEX utility to reorganize XML indexes. and the REORG TABLESPACE utility to reorganize XML table space. You do not need to specify any additional keywords in the REORG statement. When you specify that you want XML objects (either XML indexes or XML table spaces) to be reorganized, you must also specify the WORKDDN keyword and provide the specified temporary work file. The default is SYSUT1.

When you specify the name of the base table space in the REORG statement, DB2 reorganizes only that table space and not any related XML objects. If you want DB2 to reorganize the XML objects, you must specify those object names.

When you run REORG on an XML table space that supports XML versions, REORG discards rows for versions of an XML document that are no longer needed.

For XML table spaces and base table spaces with XML columns, you cannot specify the following options in the REORG statement:

- DISCARD
- REBALANCE
- UNLOAD EXTERNAL

REORG can take inline copies of XML table spaces.

If you specify a base table space with the STATISTICS keyword, DB2 does not gather statistics for the related XML table space or its indexes.

If large amounts of data are deleted from a partition-by-growth universal table space, including XML table spaces, run the REORG TABLESPACE utility with SHRLEVEL REFERENCE or SHRLEVEL CHANGE on the entire table space to reclaim physical space from the partition-by-growth and XML table spaces.

Do not use REORG UNLOAD ONLY to propagate data. When you specify the UNLOAD ONLY option, REORG unloads only the data that physically resides in the base table space; LOB and XML columns are not unloaded. For purposes of data propagation, use UNLOAD or REORG UNLOAD EXTERNAL instead.

Example 9-19 on page 216 shows the JCL for the REORG TABLESPACE utility, and the output of the utility run.

*Example 9-19 REORG TABLESPACE utility JCL and output*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SORTWK01 DD DSN=XMLR4.SORTWK01,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTWK02 DD DSN=XMLR4.SORTWK02,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTWK03 DD DSN=XMLR4.SORTWK03,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTWK04 DD DSN=XMLR4.SORTWK04,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSREC DD DSN=XMLR4.DSN00242.REORG.DATA,
//     DISP=(MOD,CATLG)
//DSNUPROC.SYSUT1 DD DSN=XMLR4.SYSUT1,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SORTOUT DD DSN=XMLR4.SORTOUT,
//     DISP=(MOD,DELETE,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSIN    DD  *
REORG TABLESPACE DSN00242.BKRTORCS
REORG TABLESPACE DSN00242.XBKR0000

1DSNU000I    319 18:06:00.66 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   319 18:06:00.68 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    319 18:06:00.69 DSNUGUTC -  REORG TABLESPACE DSN00242.BKRTORCS
 DSNU251I    319 18:06:00.81 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=1 FOR TABLESPACE
DSN00242.BKRTORCS PART 1
 DSNU252I    319 18:06:00.81 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=1 FOR TABLESPACE
DSN00242.BKRTORCS
 DSNU250I    319 18:06:00.81 DSNURULD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU303I  -DB0B 319 18:06:01.21 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=1 FOR TABLE
XMLR4.BK_TO_CSTMR_STMT PART=1
 DSNU304I  -DB0B 319 18:06:01.21 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=1 FOR TABLE
XMLR4.BK_TO_CSTMR_STMT
 DSNU302I    319 18:06:01.22 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=1
 DSNU300I    319 18:06:01.22 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU042I    319 18:06:01.22 DSNUGSOR - SORT PHASE STATISTICS -
                       NUMBER OF RECORDS=1
                       ELAPSED TIME=00:00:00
 DSNU349I  -DB0B 319 18:06:01.28 DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=1 FOR INDEX
XMLR4.I_DOCIDBK_TO_CSTMR_STMT
 DSNU258I    319 18:06:01.28 DSNURBXD - BUILD PHASE STATISTICS - NUMBER OF INDEXES=1
 DSNU259I    319 18:06:01.28 DSNURBXD - BUILD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU421I    319 18:06:01.30 DSNUGFUM - START OF DFSMS MESSAGES

1PAGE 0001     5695-DF175  DFSMSDSS V1R12.0 DATA SET SERVICES     2010.319 18:06
-ADR030I (SCH)-PRIME( 0), DCB VALUES HAVE BEEN MODIFIED FOR SYSPRINT. BLKSIZE VALUE MODIFIED FROM 0 TO 128
  COPY DATASET(INCLUDE( -
  DB0BD.DSNDBC.DSN00242.BKRTORCS.I0001.A001 )) -
  RENAMEU( -
  (DB0BD.DSNDBC.DSN00242.BKRTORCS.I0001.A001 , -
  DB0BI.DSN00242.BKRTORCS.N00001.C2RPDUCT )) -
  REPUNC ALLDATA(*) ALLEXCP CANCELERROR SHARE -
  WRITECHECK TOLERATE(ENQF)
 ADR101I (R/I)-RI01 (01), TASKID 001 HAS BEEN ASSIGNED TO COMMAND 'COPY '
 ADR109I (R/I)-RI01 (01), 2010.319 18:06:01 INITIAL SCAN OF USER CONTROL STATEMENTS COMPLETED
 ADR050I (001)-PRIME(01), DFSMSDSS INVOKED VIA APPLICATION INTERFACE
 ADR016I (001)-PRIME(01), RACF® LOGGING OPTION IN EFFECT FOR THIS TASK
```

```
0ADR006I (001)-STEND(01), 2010.319 18:06:01 EXECUTION BEGINS
0ADR711I (001)-NEWDS(01), DATA SET DB0BD.DSNDBC.DSN00242.BKRTORCS.I0001.A001 HAS BEEN ALLOCATED WITH NEWNAME
                        DB0BI.DSN00242.BKRTORCS.N00001.C2RPDUCT USING STORCLAS DB0BDATA, DATACLAS DB0B, AND
MGMTCLAS MCDB22
0ADR806I (001)-TOMI (03), DATA SET DB0BD.DSNDBC.DSN00242.BKRTORCS.I0001.A001 COPIED USING A FAST REPLICATION
FUNCTION
0ADR801I (001)-DDDS (01), DATA SET FILTERING IS COMPLETE. 1 OF 1 DATA SETS WERE SELECTED: 0 FAILED
SERIALIZATION AND 0 FAILED FOR OTHER REASONS
0ADR454I (001)-DDDS (01), THE FOLLOWING DATA SETS WERE SUCCESSFULLY PROCESSED
0                          DB0BD.DSNDBC.DSN00242.BKRTORCS.I0001.A001
0ADR006I (001)-STEND(02), 2010.319 18:06:01 EXECUTION ENDS
0ADR013I (001)-CLTSK(01), 2010.319 18:06:01 TASK COMPLETED WITH RETURN CODE 0000
0ADR012I (SCH)-DSSU (01), 2010.319 18:06:01 DFSMSDSS PROCESSING COMPLETE. HIGHEST RETURN CODE IS 0000
 DSNU422I    319 18:06:01.48 DSNUGFCD - END OF DFSMS MESSAGE

0DSNU050I    319 18:06:01.49 DSNUGUTC -  REORG TABLESPACE DSN00242.XBKR0000
 DSNU251I    319 18:06:01.58 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=1 FOR TABLESPACE
DSN00242.XBKR0000 PART 1
 DSNU252I    319 18:06:01.58 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=1 FOR TABLESPACE
DSN00242.XBKR0000
 DSNU250I    319 18:06:01.58 DSNURULD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU303I  -DB0B 319 18:06:01.97 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=1 FOR TABLE
XMLR4.XBK_TO_CSTMR_STMT PART=1
 DSNU304I  -DB0B 319 18:06:01.97 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=1 FOR TABLE
XMLR4.XBK_TO_CSTMR_STMT
 DSNU302I    319 18:06:01.97 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=1
 DSNU300I    319 18:06:01.97 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU042I    319 18:06:01.97 DSNUGSOR - SORT PHASE STATISTICS -
                       NUMBER OF RECORDS=1
                       ELAPSED TIME=00:00:00
 DSNU349I  -DB0B 319 18:06:02.06 DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=1 FOR INDEX
XMLR4.I_NODEIDXBK_TO_CSTMR_STMT
 DSNU258I    319 18:06:02.06 DSNURBXD - BUILD PHASE STATISTICS - NUMBER OF INDEXES=1
 DSNU259I    319 18:06:02.06 DSNURBXD - BUILD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU421I    319 18:06:02.08 DSNUGFUM - START OF DFSMS MESSAGES

1PAGE 0001     5695-DF175  DFSMSDSS V1R12.0 DATA SET SERVICES     2010.319 18:06
- COPY DATASET(INCLUDE( -
  DB0BD.DSNDBC.DSN00242.XBKR0000.I0001.A001 )) -
  RENAMEU( -
  (DB0BD.DSNDBC.DSN00242.XBKR0000.I0001.A001 , -
  DB0BI.DSN00242.XBKR0000.N00001.C2RPDUYM )) -
  REPUNC ALLDATA(*) ALLEXCP CANCELERROR SHARE -
  WRITECHECK TOLERATE(ENQF)
 ADR101I (R/I)-RI01 (01), TASKID 001 HAS BEEN ASSIGNED TO COMMAND 'COPY '
 ADR109I (R/I)-RI01 (01), 2010.319 18:06:02 INITIAL SCAN OF USER CONTROL STATEMENTS COMPLETED
 ADR050I (001)-PRIME(01), DFSMSDSS INVOKED VIA APPLICATION INTERFACE
 ADR016I (001)-PRIME(01), RACF LOGGING OPTION IN EFFECT FOR THIS TASK
0ADR006I (001)-STEND(01), 2010.319 18:06:02 EXECUTION BEGINS
0ADR711I (001)-NEWDS(01), DATA SET DB0BD.DSNDBC.DSN00242.XBKR0000.I0001.A001 HAS BEEN ALLOCATED WITH NEWNAME
                        DB0BI.DSN00242.XBKR0000.N00001.C2RPDUYM USING STORCLAS DB0BDATA, DATACLAS DB0B, AND
MGMTCLAS MCDB22
0ADR806I (001)-TOMI (03), DATA SET DB0BD.DSNDBC.DSN00242.XBKR0000.I0001.A001 COPIED USING A FAST REPLICATION
FUNCTION
0ADR801I (001)-DDDS (01), DATA SET FILTERING IS COMPLETE. 1 OF 1 DATA SETS WERE SELECTED: 0 FAILED
SERIALIZATION AND 0 FAILED FOR OTHER REASONS
0ADR454I (001)-DDDS (01), THE FOLLOWING DATA SETS WERE SUCCESSFULLY PROCESSED
0                          DB0BD.DSNDBC.DSN00242.XBKR0000.I0001.A001
0ADR006I (001)-STEND(02), 2010.319 18:06:02 EXECUTION ENDS
0ADR013I (001)-CLTSK(01), 2010.319 18:06:02 TASK COMPLETED WITH RETURN CODE 0000
0ADR012I (SCH)-DSSU (01), 2010.319 18:06:02 DFSMSDSS PROCESSING COMPLETE. HIGHEST RETURN CODE IS 0000
 DSNU422I    319 18:06:02.27 DSNUGFCD - END OF DFSMS MESSAGE

DSNU010I    319 18:06:02.29 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

The REORG table space utility reorganizes also the DOCID index (and any user-defined indexes) when reorganizing the base table space, and the NODEID index when reorganizing the XML table space.

You can reorganize only the indexes by using the REORG INDEX utility.

## 9.13 REPAIR

Use the REPAIR utility on XML objects to perform the following tasks:

- Set the status of an XML column to invalid.
- Delete a corrupted XML document and its NODEID index entries.

The most common use for the REPAIR utility for XML objects is to take corrective action after you run CHECK DATA with SHRLEVEL CHANGE on a table space with XML columns. CHECK DATA with SHRLEVEL CHANGE operates on shadow data sets, so it does not modify XML columns or XML table spaces. Instead, CHECK DATA generates REPAIR statements that you can run to delete invalid XML documents and to mark the corresponding XML columns as invalid.

For examples of invoking the REPAIR utility when diagnosing problems with XML data, see 10.5, “Diagnostics” on page 242.

## 9.14 REPORT

When you specify REPORT TABLESPACESET, the output report includes XML objects in the list of members in the table space set.

The sample output in Example 9-20 shows a table space set for a table that contains an XML column.

*Example 9-20 REPORT utility JCL (and TABLESPACESET option) and output*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DBOBM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DBOB,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
REPORT TABLESPACESET DSN00242.BKRTORCS

1DSNU000I    315 16:50:26.26 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   315 16:50:26.28 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    315 16:50:26.28 DSNUGUTC -  REPORT TABLESPACESET DSN00242.BKRTORCS
 DSNU587I  -DBOB 315 16:50:26.29 DSNUPSET - REPORT TABLESPACE SET WITH TABLESPACE
DSN00242.BKRTORCS

 TABLESPACE SET REPORT:

 TABLESPACE         : DSN00242.BKRTORCS
   TABLE            : XMLR4.BK_TO_CSTMR_STMT
   INDEXSPACE       : DSN00242.IRDOCIDB
   INDEX            : XMLR4.I_DOCIDBK_TO_CSTMR_STMT


 XML TABLESPACE SET REPORT:

 TABLESPACE                 : DSN00242.BKRTORCS

   BASE TABLE               : XMLR4.BK_TO_CSTMR_STMT
   COLUMN                   : BK_TO_CSTMR_STMT
     XML TABLESPACE         : DSN00242.XBKR0000
       XML TABLE            : XMLR4.XBK_TO_CSTMR_STMT
       XML NODEID INDEXSPACE: DSN00242.IRNODEID
       XML NODEID INDEX     : XMLR4.I_NODEIDXBK_TO_CSTMR_STMT

 DSNU580I    315 16:50:26.29 DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00

NOTE: The same result is produced if you use the following:
REPORT TABLESPACESET DSN00242.XBKR0000
```

When you specify REPORT RECOVERY, the output report includes recovery-related information. Use REPORT RECOVERY to find information that is necessary for recovering a table space, index, or a table space and all of its indexes. This approach is particularly useful for point-in-time recovery.

Example 9-21 shows the JCL for the REPORT utility with the RECOVERY option for the base table space and the output of the utility run.

*Example 9-21 REPORT utility JCL (and RECOVERY option for base table space) and output*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
REPORT RECOVERY TABLESPACE DSN00242.BKRTORCS

1DSNU000I    315 17:02:40.42 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   315 17:02:40.45 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    315 17:02:40.45 DSNUGUTC -  REPORT RECOVERY TABLESPACE DSN00242.BKRTORCS
 DSNU581I  -DB0B 315 17:02:40.45 DSNUPREC - REPORT RECOVERY TABLESPACE DSN00242.BKRTORCS
 DSNU593I  -DB0B 315 17:02:40.46 DSNUPREC -  REPORT RECOVERY ENVIRONMENT RECORD:
              MINIMUM   RBA: 000000000000
              MAXIMUM   RBA: FFFFFFFFFFFF
              MIGRATING RBA: 000000000000
 DSNU582I  -DB0B 315 17:02:40.46 DSNUPPCP - REPORT RECOVERY TABLESPACE DSN00242.BKRTORCS
SYSCOPY ROWS AND SYSTEM LEVEL BACKUPS
 TIMESTAMP = 2010-11-04-22.02.43.096801,  IC TYPE = *C*,  SHR LVL =  ,  DSNUM    = 0000,
START LRSN =000066569186
 DEV TYPE  =         ,                    IC BACK =   ,   STYPE    = L,  FILE SEQ = 0000,
PIT LRSN  = 000000000000
 LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,
LOGGED = Y,   TTYPE =
 JOBNAME   =         ,     AUTHID =           ,     COPYPAGESF = -1.0E+00
 NPAGESF   = -1.0E+00            ,                  CPAGESF = -1.0E+00
 DSNAME    = DSN00242.BKRTORCS                            , MEMBER NAME =         ,
INSTANCE = 01, RELCREATED = M
................................................................................
................................................................................
................................................................................
TIMESTAMP = 2010-11-10-18.12.48.246523,  IC TYPE =  F ,  SHR LVL = R,  DSNUM    = 0000,
START LRSN =000069667C5E
 DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE    =  ,  FILE SEQ = 0000,
PIT LRSN  = 000000000000
 LOW DSNUM = 0001,   HIGH DSNUM = 0001,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,
LOGGED = Y,   TTYPE =
 JOBNAME   = XMLR4LD ,     AUTHID =   XMLR4   ,     COPYPAGESF =  3.0E+00
 NPAGESF   =  3.4E+01            ,                  CPAGESF =  0.0E0
 DSNAME    = DSN00242.BKRTORCS.F.D2010314.T231248.COPY     , MEMBER NAME =         ,
INSTANCE = 01, RELCREATED = M
```

```
 TIMESTAMP = 2010-11-10-18.14.25.344136,  IC TYPE =  I ,  SHR LVL = R,  DSNUM    = 0000,
START LRSN =0000696BCB4C
 DEV TYPE  = 3390    ,                   IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,
PIT LRSN  = 000000000000
 LOW DSNUM = 0001,   HIGH DSNUM = 0001,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,
LOGGED = Y,   TTYPE =
 JOBNAME   = XMLR4LD ,     AUTHID =   XMLR4   ,     COPYPAGESF =  3.0E+00
 NPAGESF   =  3.4E+01           ,               CPAGESF =  2.0E+00
 DSNAME    = DSN00242.BKRTORCS.I.D2010314.T231425.COPY     , MEMBER NAME =        ,
INSTANCE = 01, RELCREATED = M

 TIMESTAMP = 2010-11-10-18.30.01.134384,  IC TYPE =  F ,  SHR LVL = R,  DSNUM    = 0000,
START LRSN =0000696BCB4C
 DEV TYPE  = 3390    ,                   IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,
PIT LRSN  = 000000000000
 LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,
LOGGED = Y,   TTYPE =
 JOBNAME   = XMLR4LD ,     AUTHID =   XMLR4   ,     COPYPAGESF =  3.0E+00
 NPAGESF   =  3.4E+01           ,               CPAGESF =  0.0E0
 DSNAME    = DSN00242.BKRTORCS.F.D2010314.T233001.COPY     , MEMBER NAME =        ,
INSTANCE = 01, RELCREATED = M
................................................................................
................................................................................
................................................................................
DSNU580I    315 17:02:40.46 DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00

 DSNU010I    315 17:02:40.46 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Example 9-22 shows the JCL for the REPORT utility with the RECOVERY option for the XML table space and the output of the utility run.

*Example 9-22 REPORT utility JCL (and TRECOVERY option for XML table space) and output*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DBOBM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DBOB,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
REPORT RECOVERY TABLESPACE DSN00242.XBKR0000

1DSNU000I    315 17:07:09.65 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   315 17:07:09.68 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    315 17:07:09.68 DSNUGUTC -  REPORT RECOVERY TABLESPACE DSN00242.XBKR0000
 DSNU581I  -DBOB 315 17:07:09.68 DSNUPREC - REPORT RECOVERY TABLESPACE DSN00242.XBKR0000
 DSNU593I  -DBOB 315 17:07:09.69 DSNUPREC -  REPORT RECOVERY ENVIRONMENT RECORD:
              MINIMUM   RBA: 000000000000
              MAXIMUM   RBA: FFFFFFFFFFFF
              MIGRATING RBA: 000000000000
 DSNU582I  -DBOB 315 17:07:09.69 DSNUPPCP - REPORT RECOVERY TABLESPACE DSN00242.XBKR0000
SYSCOPY ROWS AND SYSTEM LEVEL BACKUPS
TIMESTAMP = 2010-11-04-22.02.43.211430,  IC TYPE = *C*,  SHR LVL =  ,  DSNUM    = 0000,
START LRSN =00006656ED79
 DEV TYPE  =        ,                    IC BACK =   ,   STYPE   = L,  FILE SEQ = 0000,
PIT LRSN  = 000000000000
 LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,
LOGGED = Y,   TTYPE =
 JOBNAME   =        ,     AUTHID =          ,     COPYPAGESF = -1.0E+00
 NPAGESF   = -1.0E+00           ,               CPAGESF = -1.0E+00
 DSNAME    = DSN00242.XBKR0000                            , MEMBER NAME =         ,
INSTANCE = 01, RELCREATED = M
..................................................................................
..................................................................................
..................................................................................
TIMESTAMP = 2010-11-10-18.12.48.307770,  IC TYPE =  F ,  SHR LVL = R,  DSNUM    = 0000,
START LRSN =000069667C5E
 DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,
PIT LRSN  = 000000000000
 LOW DSNUM = 0001,   HIGH DSNUM = 0001,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,
LOGGED = Y,   TTYPE =
 JOBNAME   = XMLR4LD ,     AUTHID =   XMLR4   ,     COPYPAGESF =  3.0E+00
 NPAGESF   =  4.5E+01            ,                  CPAGESF =  0.0E0
 DSNAME    = DSN00242.XBKR0000.F.D2010314.T231248.COPY     , MEMBER NAME =         ,
INSTANCE = 01, RELCREATED = M

 TIMESTAMP = 2010-11-10-18.14.25.421739,  IC TYPE =  I ,  SHR LVL = R,  DSNUM    = 0000,
START LRSN =0000696BCB4C
 DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,
PIT LRSN  = 000000000000
 LOW DSNUM = 0001,   HIGH DSNUM = 0001,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,
LOGGED = Y,   TTYPE =
 JOBNAME   = XMLR4LD ,     AUTHID =   XMLR4   ,     COPYPAGESF =  3.0E+00
 NPAGESF   =  4.5E+01            ,                  CPAGESF =  2.0E+00
 DSNAME    = DSN00242.XBKR0000.I.D2010314.T231425.COPY     , MEMBER NAME =         ,
INSTANCE = 01, RELCREATED = M

 TIMESTAMP = 2010-11-10-18.30.01.235727,  IC TYPE =  F ,  SHR LVL = R,  DSNUM    = 0000,
START LRSN =0000696BCB4C
 DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,
PIT LRSN  = 000000000000
 LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,
LOGGED = Y,   TTYPE =
 JOBNAME   = XMLR4LD ,     AUTHID =   XMLR4   ,     COPYPAGESF =  3.0E+00
 NPAGESF   =  4.5E+01            ,                  CPAGESF =  0.0E0
 DSNAME    = DSN00242.XBKR0000.F.D2010314.T233001.COPY     , MEMBER NAME =         ,
INSTANCE = 01, RELCREATED = M
..................................................................................
..................................................................................
..................................................................................
DSNU580I    315 17:07:09.69 DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00
DSNU010I    315 17:07:09.70 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Although we do not show the complete output of the REPORT utility, we show only the relevant entries to demonstrate partial recovery of the table space.

We performed the following steps:

1. Took a full image copy of the base table space and XML table space (Example 9-3 on page 191).
2. Performed a partial update to the XML document (Figure 9-2 on page 192).
3. Took an incremental image copy (Example 9-4 on page 193).
4. Merged the full and incremental image copies (Example 9-13 on page 205).

The REPORT utility output shows the entries for the full, incremental, and merged full image copy for the base table space in Example 9-21 on page 219 and for the XML table space in Example 9-22 on page 220.

## 9.15 RUNSTATS

Use the RUNSTATS utility to gather statistics for XML objects. RUNSTATS TABLESPACE ignores the following keywords for XML table spaces:

- COLGROUP
- FREQVAL MOST/LEAST/BOTH
- HISTOGRAM

RUNSTATS INDEX ignores the following keywords for XML indexes or NODEID indexes:

- KEYCARD
- FREQVAL MOST/LEAST/BOTH
- HISTOGRAM

XML indexes are related to XML tables, and not to the associated base tables. If you specify a base table space and an XML index in the same RUNSTATS control statement, DB2 generates an error. When you run RUNSTATS against a base table, RUNSTATS collects statistics only for indexes on the base table, including the document ID index.

RUNSTATS TABLESPACE does not collect histogram statistics for XML table spaces. RUNSTATS INDEX does not collect histogram statistics for XML NODEID indexes or XML indexes.

Example 9-23 shows the JCL for the RUNSTATS utility for the base table space (and XML table space) and the output of the utility run.

*Example 9-23 RUNSTATS utility JCL and output*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DBOBM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DBOB,UID='TEMP',UTPROC=''
//DSNUPROC.SYSIN    DD  *
RUNSTATS TABLESPACE DSN00242.BKRTORCS INDEX (ALL)
RUNSTATS TABLESPACE DSN00242.XBKR0000 INDEX (ALL)

1DSNU000I    319 19:06:31.28 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   319 19:06:31.33 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    319 19:06:31.33 DSNUGUTC -  RUNSTATS TABLESPACE DSN00242.BKRTORCS INDEX(ALL)
 DSNU610I  -DBOB 319 19:06:31.40 DSNUSUTP - SYSTABLEPART CATALOG UPDATE FOR
DSN00242.BKRTORCS SUCCESSFUL
 DSNU610I  -DBOB 319 19:06:31.40 DSNUSUPT - SYSTABSTATS CATALOG UPDATE FOR
XMLR4.BK_TO_CSTMR_STMT SUCCESSFUL
 DSNU610I  -DBOB 319 19:06:31.40 DSNUSUTB - SYSTABLES CATALOG UPDATE FOR
XMLR4.BK_TO_CSTMR_STMT SUCCESSFUL
 DSNU610I  -DBOB 319 19:06:31.40 DSNUSUTS - SYSTABLESPACE CATALOG UPDATE FOR
DSN00242.BKRTORCS SUCCESSFUL
 DSNU610I  -DBOB 319 19:06:31.41 DSNUSUIP - SYSINDEXPART CATALOG UPDATE FOR
XMLR4.I_DOCIDBK_TO_CSTMR_STMT SUCCESSFUL
 DSNU610I  -DBOB 319 19:06:31.41 DSNUSUCO - SYSCOLUMNS CATALOG UPDATE FOR
XMLR4.I_DOCIDBK_TO_CSTMR_STMT SUCCESSFUL
 DSNU610I  -DBOB 319 19:06:31.41 DSNUSUIX - SYSINDEXES CATALOG UPDATE FOR
XMLR4.I_DOCIDBK_TO_CSTMR_STMT SUCCESSFUL
 DSNU610I  -DBOB 319 19:06:31.41 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR
XMLR4.I_DOCIDBK_TO_CSTMR_STMT SUCCESSFUL
 DSNU620I  -DBOB 319 19:06:31.41 DSNUSEOF - RUNSTATS CATALOG TIMESTAMP =
2010-11-15-19.06.31.341899
```

```
0DSNU050I    319 19:06:31.42 DSNUGUTC -  RUNSTATS TABLESPACE DSN00242.XBKR0000 INDEX(ALL)
 DSNU610I  -DB0B 319 19:06:31.48 DSNUSUTP - SYSTABLEPART CATALOG UPDATE FOR
DSN00242.XBKR0000 SUCCESSFUL
 DSNU610I  -DB0B 319 19:06:31.48 DSNUSUPT - SYSTABSTATS CATALOG UPDATE FOR
XMLR4.XBK_TO_CSTMR_STMT SUCCESSFUL
 DSNU610I  -DB0B 319 19:06:31.48 DSNUSUTB - SYSTABLES CATALOG UPDATE FOR
XMLR4.XBK_TO_CSTMR_STMT SUCCESSFUL
 DSNU610I  -DB0B 319 19:06:31.48 DSNUSUTS - SYSTABLESPACE CATALOG UPDATE FOR
DSN00242.XBKR0000 SUCCESSFUL
 DSNU610I  -DB0B 319 19:06:31.49 DSNUSUIP - SYSINDEXPART CATALOG UPDATE FOR
XMLR4.I_NODEIDXBK_TO_CSTMR_STMT SUCCESSFUL
 DSNU610I  -DB0B 319 19:06:31.49 DSNUSUKT - SYSKEYTARGETS CATALOG UPDATE FOR
XMLR4.I_NODEIDXBK_TO_CSTMR_STMT SUCCESSFUL
 DSNU610I  -DB0B 319 19:06:31.49 DSNUSUIX - SYSINDEXES CATALOG UPDATE FOR
XMLR4.I_NODEIDXBK_TO_CSTMR_STMT SUCCESSFUL
 DSNU620I  -DB0B 319 19:06:31.49 DSNUSEOF - RUNSTATS CATALOG TIMESTAMP =
2010-11-15-19.06.31.425894

DSNU010I    319 19:06:31.50 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

## 9.16 UNLOAD

You can unload XML data to output data records or to separate files. Use either of the following methods to XML columns can be unloaded by using either of the following methods:

- The XML column can be unloaded to the output records. XML column value can be placed in the OUTPUT record with or without any other unloading column values. The output record can be in non-delimited or delimited format:
  - For a non-delimited format, the XML column is handled like a variable character field with a 2-byte length preceding the XML value.
  - For a delimited format, no byte length is present. If the total output record length is more than 32 KB, unload the record in spanned record format by specifying the SPANNED YES option.
- The XML column can be unloaded to a separate file whether the XML column length is less than 32 KB or not.

The output data can be in the textual XML format or the binary XML format. Data that is unloaded can be in the non-delimited or delimited format.

To unload XML data directly to output record, specify XML as the output field type. If the output is a non-delimited format, a 2-byte length precedes the value of the XML. For delimited output, no length field is present. XML is the only acceptable field type when unloading the XML directly to the output record. No data type conversion applies and you cannot specify FROMCOPY.

If the input data is in Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format (binary XML format), you need to specify BINARYXML. To unload XML data to a separate file:

- In the UNLOAD utility control statement, specify BLOBF, CLOBF or DBCLOBF. These keywords indicate that the output column contains the name of a file to which the XML value is to be unloaded. Also specify either CHAR or VARCHAR instead of XML. Do not specify FROMCOPY.
- Use the template control statement to create the XML output file and filename. If data sets are not created and the DSN type is not specified on the template, UNLOAD will use PDS

as the data set type. PDS has a limit of single volume. The output file uses multiple volumes, so you must specify HFS as the DSN type.

In the UNLOAD statement, specify the base table space. You cannot specify the XML table space.

Example 9-24 shows the JCL for the UNLOAD utility and the output of the utility run.

*Example 9-24 UNLOAD utility JCL and output*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SYSREC DD DSN=XMLR4.UNLOAD
//     DISP=(MOD,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSPUNCH DD DSN=XMLR4.LOADCTL
//     DISP=(MOD,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSIN    DD  *
UNLOAD TABLESPACE DSN00242.BKRTORCS
  FROM TABLE XMLR4.BK_TO_CSTMR_STMT
      (BK_TO_CSTMR_STMT POSITION(*) XML)

1DSNU000I    301 18:01:07.19 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   301 18:01:07.21 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    301 18:01:07.22 DSNUGUTC -  UNLOAD TABLESPACE DSN00242.BKRTORCS
 DSNU650I  -DB0B 301 18:01:07.22 DSNUUGMS -   FROM TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU650I  -DB0B 301 18:01:07.22 DSNUUGMS -    (BK_TO_CSTMR_STMT POSITION(*) XML)
 DSNU253I    301 18:01:07.24 DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
UNLOADED=1 FOR TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU252I    301 18:01:07.24 DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
UNLOADED=1 FOR TABLESPACE DSN00242.BKRTORCS
 DSNU250I    301 18:01:07.24 DSNUUNLD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU010I    301 18:01:07.25 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

NOTE: SYSREC shows the name of the data set which receives the XML document unloaded by
UNLOAD utility.
SYSPUNCH shows the name of the data set which has the LOAD utility control statements
generated by UNLOAD utility.
The first few characters of the XML document in SYSREC data set are shown below:
    7<?xml version="1.0" encoding="IBM037"?><Document xmlns:xsi="http://
0001F46A994A89A89977F4F748989889877CCDFFF7664C98A989A4A999A7AA8778AA9766
03017CF74305592965EF1B0F055364957EF924037FFEC46344553074352A729EF8337A11
A 2-byte length precedes the XML value.
The LOAD utility control statement in SYSPUNCH data set is shown below:
LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 EBCDIC  CCSID(00037,00000,00000)
 INTO TABLE "XMLR4"."BK_TO_CSTMR_STMT"
 WHEN(00001:00002) = X'0003'
 NUMRECS 1
( "BK_TO_CSTMR_STMT"
 POSITION(  *)           XML PRESERVE WHITESPACE
)
```

### UNLOAD utility using file reference variable

Example 9-25 shows the JCL for the UNLOAD utility, using file reference variable, and the output of the utility run.

*Example 9-25 UNLOAD utility JCL (using file reference variable) and output*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SYSREC DD DSN=XMLR4.UNLOAD1,
//     DISP=(MOD,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSPUNCH DD DSN=XMLR4.LOADCTL1,
//     DISP=(MOD,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSIN    DD  *
TEMPLATE TCLOBF UNIT(SYSDA) DISP(MOD,CATLG,DELETE)
DSN(&USERID..&DB..&TS..T&TI..UFILEREF)
UNLOAD TABLESPACE DSN00242.BKRTORCS
     FROM TABLE XMLR4.BK_TO_CSTMR_STMT
      (BK_TO_CSTMR_STMT POSITION(*) VARCHAR CLOBF TCLOBF)

1DSNU000I    301 19:12:35.80 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   301 19:12:35.82 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    301 19:12:35.83 DSNUGUTC -  TEMPLATE TCLOBF UNIT(SYSDA) DISP(MOD,
CATLG, DELETE) DSN(&USERID..&DB..&TS..T&TI..UFILEREF)
 DSNU1035I   301 19:12:35.83 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    301 19:12:35.83 DSNUGUTC -  UNLOAD TABLESPACE DSN00242.BKRTORCS
 DSNU650I  -DB0B 301 19:12:35.83 DSNUUGMS -   FROM TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU650I  -DB0B 301 19:12:35.83 DSNUUGMS -    (BK_TO_CSTMR_STMT POSITION(*)
VARCHAR CLOBF TCLOBF)
 DSNU1038I   301 19:12:35.89 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=TCLOBF
                        DDNAME=SYS00001
                        DSN=XMLR4.DSN00242.XBKR0000.T231235.UFILEREF
 DSNU253I    301 19:12:35.92 DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF
RECORDS UNLOADED=1 FOR TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU252I    301 19:12:35.92 DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF
RECORDS UNLOADED=1 FOR TABLESPACE DSN00242.BKRTORCS
 DSNU250I    301 19:12:35.93 DSNUUNLD - UNLOAD PHASE COMPLETE, ELAPSED
TIME=00:00:00
 DSNU010I   301 19:12:35.94 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN
CODE=0

NOTE: SYSREC shows the name of the data set derived from the Template definition
which has the XML document as shown below:
XMLR4.DSN00242.XBKR0000.T231235.UFILEREF(C1XSJ2ZY)
The first few characters of the XML document are shown below:
<?xml version="1.0" encoding="IBM037"?><Document xmlns:xsi="http://www
46A994A89A89977F4F748989889877CCDFFF7664C98A989A4A999A7AA8778AA9766AAA
CF74305592965EF1B0F055364957EF924037FFEC46344553074352A729EF8337A1166A
The XML document starts from position 1.
The LOAD utility control statement in SYSPUNCH data set is shown below:
```

```
LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 EBCDIC  CCSID(00037,00000,00000)
 INTO TABLE
 "XMLR4"."BK_TO_CSTMR_STMT"
 WHEN(00001:00002) = X'0003'
 NUMRECS 1
( "BK_TO_CSTMR_STMT"
 POSITION(  00003:00259) VARCHAR  CLOBF         PRESERVE WHITESPACE
)
```

### UNLOAD utility to unload XML data in binary

Example 9-26 shows the JCL for the UNLOAD utility to unload XML data in binary and the output of the utility run.

*Example 9-26 UNLOAD utility JCL (to unload XML data in binary) and output*

```
//XMLR4LD  JOB (999,POK),'DB0B',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DB0B,UID='TEMP',UTPROC=''
//DSNUPROC.SYSREC DD DSN=XMLR4.UNLOAD.XML.BINARY.REC00,
//     DISP=(MOD,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSPUNCH DD DSN=XMLR4.UNLOAD.XML.BINARY.PUNCH,
//     DISP=(MOD,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSIN    DD  *
UNLOAD DATA
FROM TABLE XMLR4.BK_TO_CSTMR_STMT
 (BK_TO_CSTMR_STMT XML BINARYXML)

1DSNU000I    308 14:17:16.66 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   308 14:17:16.69 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    308 14:17:16.69 DSNUGUTC -  UNLOAD DATA
 DSNU650I  -DB0B 308 14:17:16.69 DSNUUGMS -   FROM TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU650I  -DB0B 308 14:17:16.69 DSNUUGMS -    (BK_TO_CSTMR_STMT XML BINARYXML)
 DSNU253I    308 14:17:16.76 DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF
RECORDS UNLOADED=1 FOR TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU252I    308 14:17:16.76 DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF
RECORDS UNLOADED=1 FOR TABLESPACE DSN00242.BKRTORCS
 DSNU250I    308 14:17:16.77 DSNUUNLD - UNLOAD PHASE COMPLETE, ELAPSED
TIME=00:00:00

NOTE: SYSREC shows the name of the data set which receives the XML document
unloaded by UNLOAD utility.
SYSPUNCH shows the name of the data set which has the LOAD utility control
statements generated by UNLOAD utility.
The first few characters of the XML document in SYSREC data set are shown below:
....-.......ñ.ÌËÑ.ñ.ÇÈÈø...ÏÏÏ.Ï..?ÊÅ......ì(<ëÄÇÁ_/.Ñ>ËÈ/>ÄÁÃñ.ÍÊ>.ÑË?.ËÈÀ.ÑË?.
0003C300000040776142677732277727326762333325445666662667766666427763676377636763
03D4AB5100029383989984440AFF777E73EF27F2001F8DC3385D1D9E341E3569E52EA93FA344A93FA
A 2-byte length precedes the XML value.
```

```
The LOAD utility control statement in SYSPUNCH data set is shown below:
LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 EBCDIC  CCSID(00037,00000,00000)
 INTO TABLE "XMLR4"."BK_TO_CSTMR_STMT"
 WHEN(00001:00002) = X'0003'
 NUMRECS                   1
 ( "BK_TO_CSTMR_STMT"
  POSITION(  *)          XML PRESERVE WHITESPACE BINARYXML)
```

## UNLOAD utility to unload into a VBS data set in spanned record format

To unload data from a table that has large LOB or XML fields, consider unloading the data in spanned record format to improve performance of read-write operations. When you unload data in spanned record format, all LOB and XML data for a given table space or table space partition can be written to an individual sequential file. This file can reside on DASD and can span multiple volumes. Having such a single sequential file can improve the performance of read-write operations.

To unload data in spanned record format, specify the SPANNED YES option. Specify in the field specification list that all LOB and XML data are to be at the end of the record.

Example 9-27 shows the JCL for the UNLOAD utility to unload XML data in spanned record format and the output of the utility run.

*Example 9-27   UNLOAD utility JCL (to unload XML data in spanned record format) and output*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=XMLR4,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DBOBM.PROCLIB)
//UTIL EXEC DSNUPROC,SYSTEM=DBOB,UID='TEMP',UTPROC=''
//DSNUPROC.SYSREC DD DSN=XMLR4.DSN00242.XBKR0000.T214620.UFILEREF.NEW,
//     DISP=(MOD,CATLG)
//DSNUPROC.SYSPUNCH DD DSN=XMLR4.UNLOAD.SCENARIO.NEW.PUNCH3,
//     DISP=(MOD,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSIN    DD  *
UNLOAD DATA SPANNED YES FROM TABLE XMLR4.BK_TO_CSTMR_STMT
      (MSG_ID          VARCHAR,
       MSG_CRE_DT_TM    TIMESTAMP WITH TIME ZONE EXTERNAL,
       BK_TO_CSTMR_STMT XML)

NOTE: The record format of the SYSREC data set is VBS.
1DSNU000I    309 18:59:48.56 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   309 18:59:48.58 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    309 18:59:48.59 DSNUGUTC -  UNLOAD DATA SPANNED YES
 DSNU650I  -DBOB 309 18:59:48.59 DSNUUGMS -   FROM TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU650I  -DBOB 309 18:59:48.59 DSNUUGMS -    (MSG_ID VARCHAR,
 DSNU650I  -DBOB 309 18:59:48.59 DSNUUGMS -     MSG_CRE_DT_TM TIMESTAMP WITH TIME ZONE
EXTERNAL,
 DSNU650I  -DBOB 309 18:59:48.59 DSNUUGMS -     BK_TO_CSTMR_STMT XML)
 DSNU253I    309 18:59:48.65 DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
UNLOADED=1 FOR TABLE XMLR4.BK_TO_CSTMR_STMT
 DSNU252I    309 18:59:48.65 DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
UNLOADED=1 FOR TABLESPACE DSN00242.BKRTORCS
 DSNU250I    309 18:59:48.66 DSNUUNLD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNU010I    309 18:59:48.66 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

From a performance perspective, HFS is generally much better than PDS for file references. Binary XML is significantly better than textual XML. The best performing combination is SPANNED with BINARYXML.

## 9.17 DSNTIAUL

Like the UNLOAD utility, the DSNTIAUL sample program also provides two ways to handle XML data:

- Unload XML columns as normal data columns to the `SYSRECxx` file.
- Unload XML columns to a separate file.

### Unload LOB data as normal data columns (SQL parameter)

Because the maximum record length of a sequential file in z/OS is 32 KB, this method can be used only if the total record size of the data to be unloaded does not exceed 32 KB. The XML fields are unloaded together with the other selected data fields to the output file with a maximum record length of 32 KB.

In most cases, this method is used only if the XML documents in the table are small.

We demonstrate this with the XMLR4.BK_TO_CSTMR_STMT table, defined in Example 4-1 on page 54.

In the first case, we execute the JCL as shown in Example 9-28.

*Example 9-28 DSNTIAUL with SQL parameter*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=&SYSUID,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DBOBM.PROCLIB)
//JOBLIB  DD  DSN=DBOBT.SDSNLOAD,DISP=SHR
//DSNTIAUL EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DBOB)
RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB10) PARMS('SQL') -
LIB('DBOBM.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSREC00 DD DSN=XMLR4.DBOB.DSN8UNLD.SQL.SYSREC00,
// DISP=(MOD,CATLG),UNIT=3390,
// SPACE=(CYL,(100,100))
//SYSPUNCH DD DSN=XMLR4.DBOB.DSN8UNLD.SQL.SYSPUNCH,
// DISP=(MOD,CATLG),UNIT=3390,
// SPACE=(CYL,(1,1))
//SYSIN DD *
SELECT BK_TO_CSTMR_STMT
FROM XMLR4.BK_TO_CSTMR_STMT ;
As a result, we get an FB sequential data set SYSREC00 with a LRECL and BLKSIZE of 32,753
bytes with BK_TO_CSTMR_STMT beginning in position 1.
DSNT490I SAMPLE DATA UNLOAD PROGRAM
DSNT505I DSNTIAUL OPTIONS USED: SQL
DSNT503I UNLOAD DATA SET SYSPUNCH RECORD LENGTH SET TO    80
DSNT504I UNLOAD DATA SET SYSPUNCH BLOCK SIZE SET TO 27920
DSNT506I INPUT STATEMENT WAS NOT A FULL SELECT ON A SINGLE TABLE. LOAD STATEMENT WILL NEED
MODIFICATION.
DSNT503I UNLOAD DATA SET SYSREC00 RECORD LENGTH SET TO 32760
```

```
DSNT504I UNLOAD DATA SET SYSREC00 BLOCK SIZE SET TO 32760
DSNT495I SUCCESSFUL UNLOAD          1 ROWS OF TABLE TBLNAME
The job ends with a return code of RC=4.

NOTE:
The first few characters in the unloaded SYSREC00 data set are shown below:
...Ü<?xml version="1.0" encoding="IBM037"?><Document xmlns:xsi="http://www.w3.or
001F46A994A89A89977F4F748989889877CCDFFF7664C98A989A4A999A7AA8778AA9766AAA4AF499
001CCF74305592965EF1B0F055364957EF924037FFEC46344553074352A729EF8337A11666B63B69
DSNTIAUL generates in the SYSPUNCH data set the following LOAD control statement:
LOAD DATA LOG NO INDDN SYSREC00 INTO TABLE TBLNAME
(BK_TO_CSTMR_STMT    POSITION( 1 ) CLOB )

In the SYSIN DD * if you specify all the column names explicitly as shown below:
SELECT DB2_GENERATED_DOCID_FOR_XML,
       MSG_ID,
       MSG_CRE_DT_TM,
       BK_TO_CSTMR_STMT
FROM XMLR4.BK_TO_CSTMR_STMT ;
DSNTIAUL issues the messages as above including DSNT506I and generates in the SYSPUNCH data
set the following LOAD control statement:
LOAD DATA LOG NO INDDN SYSREC00 INTO TABLE TBLNAME
(DB2_GENERATED_DOCID_FOR_XML POSITION( 1 ) BIGINT  NULLIF( 9)='?',
 MSG_ID                      POSITION( 10 ) VARCHAR  NULLIF(47)='?',
 MSG_CRE_DT_TM    POSITION( 48 ) TIMESTAMP WITH TIME ZONE EXTERNAL(32)
      NULLIF(80)='?',
 BK_TO_CSTMR_STMT         POSITION( 81 ) CLOB )

You should edit the LOAD control statement by replacing TBLNAME by the actual table name,
CLOB by XML and POSITION ( 1 ) by POSITION ( 3 ) or
POSITION ( 81 ) by POSITION ( 83 ) depending on how you specify the SELECT statement in
DSNTIAUL, include either RESUME YES or REPLACE, and change
INDDN SYSREC00 to INDDN SYSREC.

In the SYSIN DD * if you specify SELECT * FROM XMLR4.BK_TO_CSTMR_STMT ;
DSNTIAUL issues the messages as above excluding DSNT506I, the job ends with a return code
of RC=0, and generates in the SYSPUNCH data set the following LOAD control statement:
LOAD DATA LOG NO INDDN SYSREC00 INTO TABLE BK_TO_CSTMR_STMT
(MSG_ID                POSITION( 1 ) VARCHAR  NULLIF(38)='?',
 MSG_CRE_DT_TM    POSITION( 39 ) TIMESTAMP WITH TIME ZONE EXTERNAL(32)
      NULLIF(71)='?',
 BK_TO_CSTMR_STMT         POSITION( 72 ) CLOB )
Notice the DB2_GENERATED_DOCID_FOR_XML column is not included.

You should edit the LOAD control statement by replacing CLOB by XML and
POSITION ( 72 ) by POSITION ( 74 ), include either RESUME YES or REPLACE, and change INDDN
SYSREC00 to INDDN SYSREC.
```

## Unload XML data to a separate file (LOBFILE parameter)

This method is the preferred method, introduced in DB2 9 for z/OS. With this method, the XML values are unloaded to a different file than the normal SYSRECxx unload files. DSNTIAUL dynamically creates a sequential data set for each XML document to be unloaded. The name of the separate file is stored in the normal SYSRECxx unload-files together with the other normal data fields.

DSNTIAUL does this step by using the new XML file reference variables introduced in DB2 9. Each XML file has a name of the form `<prefix>.Q<i>.C<j>.R<k>`, where:

- `<prefix>` is a user-specified data set name prefix. It must conform to the rules for a z/OS physical sequential data set name and cannot exceed 17 characters.
- `Q<i>` is the (<i>-1) query that is processed by the current DSNTIAUL session. `<i>` ranges from 0,000,000 to 0,000,099, which corresponds to the limit on the number of queries that can be processed by a single DSNTIAUL session.
- c<j> is the (<j>-1) column in the current SELECT statement. `<j>` ranges from 0,000,000 to 0,000,999 (no more than 750 columns are permitted in a table or view).
- `R<k>` is the (<k>-1) row fetched for the current SELECT statement. `<k>` ranges from 0,000,000 to 9,999,999.

The data set prefix <prefix> is specified by means of a new DSNTIAUL run-time parameter named LOBFILE. This parameter is the same one that is used when handling LOBs.

A demonstration with the JCL is shown in Example 9-29.

*Example 9-29 DSNTIAUL with LOBFILE parameter*

```
//XMLR4LD  JOB (999,POK),'DBOB',CLASS=A,
// MSGCLASS=T,NOTIFY=&SYSUID,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB0BM.PROCLIB)
//JOBLIB  DD  DSN=DB0BT.SDSNLOAD,DISP=SHR
//DSNTIAUL EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DB0B)
RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB10) PARMS('LOBFILE(XMLR4)') -
LIB('DB0BM.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSREC00 DD DSN=XMLR4.DB0B.DSN8UNLD.XML.SYSREC00,
// DISP=(MOD,CATLG),UNIT=3390,
// SPACE=(CYL,(1,1))
//SYSPUNCH DD DSN=XMLR4.DB0B.DSN8UNLD.XML.SYSPUNCH,
// DISP=(MOD,CATLG),UNIT=3390,
// SPACE=(CYL,(1,1))
//SYSIN DD *
     XMLR4.BK_TO_CSTMR_STMT

NOTE: In this case, the SYSIN input file contains the name of the base table.

DSNT490I SAMPLE DATA UNLOAD PROGRAM
DSNT505I DSNTIAUL OPTIONS USED: LOBFILE(XMLR4)
DSNT503I UNLOAD DATA SET SYSPUNCH RECORD LENGTH SET TO    80
DSNT504I UNLOAD DATA SET SYSPUNCH BLOCK SIZE SET TO 27920
DSNT503I UNLOAD DATA SET SYSREC00 RECORD LENGTH SET TO   115
DSNT504I UNLOAD DATA SET SYSREC00 BLOCK SIZE SET TO 27945
DSNT495I SUCCESSFUL UNLOAD            1 ROWS OF TABLE XMLR4.BK_TO_CSTMR_STMT

The first few characters in the unloaded SYSREC00 data set are shown below:

..?................................?XMLR4.Q0000000.C0000002.R0000000............
006000000000000000000000000000000006EDDDF4DFFFFFFF4CFFFFFFF4DFFFFFFF000000000000
00F00000000000000000000000000000000F74394B80000000B30000002B90000000000000000000
Notice this data set has only the file name where the XML document is unloaded.
The first few characters in the data set XMLR4.Q0000000.C0000002.R0000000 are:
<?xml version="1.0" encoding="IBM037"?><Document xmlns:xsi="http://www.w3.org/20
```

```
DSNTIAUL generates in the SYSPUNCH data set the following LOAD control statement:
LOAD DATA LOG NO INDDN SYSREC00 INTO TABLE BK_TO_CSTMR_STMT
(MSG_ID                POSITION( 1 ) VARCHAR  NULLIF(38)='?',
  MSG_CRE_DT_TM    POSITION( 39 ) TIMESTAMP WITH TIME ZONE EXTERNAL(32)
      NULLIF(71)='?',
 BK_TO_CSTMR_STMT         POSITION( 72 ) CLOBF )
Notice the DB2_GENERATED_DOCID_FOR_XML column is not included.

You should edit the LOAD control statement to include either RESUME YES or REPLACE, and
change INDDN SYSREC00 to INDDN SYSREC.
```

You may also perform the following tasks:

- specify SQL as first PARM:

  ```
  RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB91) PARMS('SQL,LOBFILE(XMLR4)') -
  ```

- Use an SQL statement as input to SYSIN:

  ```
  SELECT BK_TO_CSTMR_STMT FROM XMLR4.BK_TO_CSTMR_STMT ;
  ```

In both cases, a sequential data set SYSREC00 containing the normal data fields and the names of the XML output files is produced. All the XML output files have a name in the following form (where xxxx can be in the range 0000 - 5882):

```
XMLR4.Q0000000.C0000006.R000xxxx
```

The files are dynamically allocated as sequential files with the values:

```
RECFM=VB,LRECL=27994,BLKSIZE=27998
```

This BLKSIZE is the optimal for 3390 devices.

## 9.18 DSN1COPY

Use DSN1COPY to copy tables from one subsystem to another. When you copy tables from one subsystem to another, you must ensure that the version information on the target subsystem matches the version information on the source subsystem.

> **Restriction:** DB2 XML data is condensed by substituting strings by unique IDs. These unique IDs are stored in the catalog table SYSIBM.SYSXMLSTRINGS and they are not available in the related XML table space. Therefore, do not copy XML table spaces from one subsystem to another using DSN1COPY.

10

# XML-related tasks for the DBA

This chapter, for database administrators (DBA), contains an overview of typical DBA tasks that are affected by XML. The tasks are grouped into the following sections:

- Tasks regarding system setup
- Tasks regarding object creation
- Housekeeping
- Backup and recovery
- Diagnostics

For many of the DBA tasks that are performed with utilities, we include only a brief overview. Utilities that have additional considerations, with the advent of pureXML, are covered in Chapter 9, “Utilities with XML” on page 183.

© Copyright IBM Corp. 2011. All rights reserved.

## 10.1 Tasks regarding system setup

Before creating and using XML objects, you might have to perform preliminary configuration and setup steps, such as set up XML schema repository (XSR) for XML validation, size and allocate a dedicated XML buffer pool, and size and adjust the amount of memory available for XML processing.

### 10.1.1 Setting up the XSR

Before performing XML schema validation of your XML documents, you must set up the XSR. This task involves creating a set of DB2 tables and indexes that store XML schema information, and a set of stored procedures that operate on the XML schemas, which are stored in the tables.

The major steps to set up the XSR are as follows:

1. Define the XSR tables and indexes.

   The DSNTIJRT installation job invokes a program that executes the CREATE DATABASE, CREATE TABLESPACE, CREATE TABLE, and CREATE INDEX statements for the XML schema repository tables and indexes. After the installation process customizes the DSNTIJRT job, you can run DSNTIJRT without further modification to create those tables and indexes.

   > **Important:** Do not drop these objects after you begin to do XML schema validation. Doing so can cause unexpected behavior.

2. Define the WLM environment and startup procedure for the C language XSR stored procedures.
3. Define the WLM environment and startup procedure for the Java language XSR stored procedures.

   > **Note:** The Java stored procedure XSR_COMPLETE must run non-APF authorized. This step can be accomplished by adding a non-APF-authorized data set to the STEPLIB concatenation of the WLM procedure.
   >
   > However, be aware that other Java stored procedures might need to run APF-authorized, so you might have to create a special WLM procedure for XSR_COMPLETE.

4. Define the XML schema repository stored procedures to DB2.
5. Bind the packages for the XML schema repository stored procedures.

   Installation job DSNTIJRT invokes a program that binds the packages for the XML schema repository stored procedures. After the installation process customizes the DSNTIJRT job, you can run DSNTIJRT without further modification to bind the packages.

6. Bind the packages for the IBM Data Server Driver for JDBC and SQLJ.
7. Test the XML schema repository setup.

For detailed information, see “Setting up the XML schema repository” in *DB2 10 for z/OS pureXML Guide,* SC19-2981.

### 10.1.2 Buffer pool for XML

When you create a table with an XML column, or alter a table to add an XML column, DB2 creates the XML table space and indexes implicitly. The buffer pool that is used for XML table spaces is always 16 KB.

The DEFAULT BUFFER POOL FOR USER XML DATA field (TBSBPXML subsystem parameter) specifies the default buffer pool that is to be used for XML table spaces. The default is BP16K0.

You may alter the BUFFERPOOL property of the XML table space, which supports the altering to other 16-KB buffer pools only.

### 10.1.3 Sizing XMLVALA and XMLVALS

Use the XMLVALA and XMLVALS subsystem parameters to limit the amount of DB2 virtual storage that is used for XML processing. Because XML values are not fixed in length, and can be very large, DB2 cannot estimate the amount of memory that it needs for processing SQL/XML and XPath queries before run time. DB2 allocates virtual storage at run time based on the size of the XML data. For large XML data, the amount of virtual storage that DB2 requires can grow very large.

If your DB2 subsystem encounters storage constraints because XML values are using too much memory, set the XMLVALA and XMLVALS subsystem parameters:

- To specify the maximum amount of memory, in KB, for storing XML values for each user, set XMLVALA. The default is 204800 KB.
- To specify the maximum amount of memory, in MB, for storing XML values for the entire subsystem, set XMLVALS. The default is 10240 MB.

### 10.1.4 Be up to date with maintenance

In DB2 9, many enhancements have been introduced through APARs, such as XML index for joining and several XMLTABLE performance enhancements. Apply these APARs and keep your DB2 system up-to-date to benefit from these and future enhancements.

Because DB2 use XML System Services for parsing and validating XML documents, be attentive to any maintenance offered in this area.

See information APAR II14426 for XML service.

## 10.2 Tasks regarding object creation

When you create tables with XML columns, or add XML columns to existing tables, all the XML objects are created implicitly without any action from the DBA.

However, be aware of the choices that are made for these objects. Certain properties might be inherited directly from the base table, some might be inferred from properties of the base table, and others might be dependant on default values or values of DSNZPARMs.

A good practice is to understand how these properties are derived, so you can plan them or alter them after the objects have been created. This section describes the most important properties.

## 10.2.1 Creation of table with XML columns

You may create a table with XML columns, or alter a table to add one or more XML column. When a table with an XML column is created, an XML table space, document ID index, and NODEID index are implicitly created.

For more information about creation of tables with XML columns and the storage structure for XML data, see Chapter 4, “Creating and adding XML data” on page 53.

## 10.2.2 Alteration of implicitly created XML objects

After creating or adding an XML column, you may alter implicitly created XML objects. However, you may alter only certain properties for the XML objects as listed in Table 10-1.

*Table 10-1 Properties can be altered for XML objects*

| Objects | Description |
|---|---|
| XML table space | You can alter the following properties:<br>▸ BUFFERPOOL (16 KB buffer pools only)<br>▸ COMPRESS<br>▸ PRIQTY<br>▸ SECQTY<br>▸ MAXROWS<br>▸ FREEPAGE<br>▸ PCTFREE<br>▸ GBPCACHE<br>▸ USING STOGROUP<br>▸ ERASE<br>▸ LOCKSIZE (The only possible values are XML and TABLESPACE.)<br>▸ SEGSIZE<br>▸ DSSIZE<br>▸ MAXPARTITIONS<br>XML table space attributes that are inherited from the base table space, such as LOG, are implicitly altered if the base table space is altered. |
| XML table | The ALTER TABLE ALTER PARTITION statement is not supported if the table contains an XML column. |
| Index | You cannot alter the following properties:<br>▸ CLUSTER<br>▸ PADDED<br>▸ ADD COLUMN |

## 10.2.3 Sizing table spaces

Pay special consideration to sizing table spaces when you deal with XML data in range-partitioned table spaces. Recall that the type of the base table space dictates the type of the implicitly created XML table space. The correspondence is listed in Table 10-2.

*Table 10-2 Table space types for base and XML tables*

| Base table space | XML table space |
|---|---|
| Simple | Partition-by-growth |
| Segmented | Partition-by-growth |
| Partitioned | Range-partitioned |

| Base table space | XML table space |
|---|---|
| Partition-by-growth | Partition-by-growth |
| Range-partitioned | Range-partitioned |

For partition-by-growth XML table spaces, there is no correspondence between the partition in which a particular XML document resides, and the partition of the base table row. The table spaces grow as needed, and independently of each other.

For partitioned and range-partitioned table spaces, the XML document and the base row must reside in corresponding partitions. If the base table row moves partition, so does the XML document. Therefore, the number of rows fitting into a relational partition is limited by the number of rows that fit into the XML partition.

The DSSIZE of the XML table space is dictated by a combination of the DSSIZE and page size of the base table. The exact values are listed in Table 10-3.

*Table 10-3 DSSIZE of the XML table space*

| DSSIZE of base table space | Page size 4 KB | Page size 8 KB | Page size 16 KB | Page size 32 KB |
|---|---|---|---|---|
| 1 - 4 GB | 4 GB | 4 GB | 4 GB | 4 GB |
| 8 GB | 32 GB | 16 GB | 16 GB | 16 GB |
| 16 GB | 64 GB | 32 GB | 16 GB | 16 GB |
| 32 GB | 64 GB | 64 GB | 32 GB | 16 GB |
| 64 GB | 64 GB | 64 GB | 64 GB | 64 GB |

To understand what impact the decisions made with respect to size and range of partitions, let us assume that we want to implement the BK_TO_CSTMR_STMT table as a range-partitioned table with one partition per year. The DDL to create this table is shown in Example 10-1.

*Example 10-1 Creating a range-partitioned table*

```
CREATE TABLESPACE BKSTTS01
     IN BKSTDB01
     USING STOGROUP SYSDEFLT
     DSSIZE 4G
     BUFFERPOOL BP0
     NUMPARTS 3#
CREATE TABLE XMLR2.BK_TO_CSTMR_STMT
   (MSG_ID              VARCHAR(35) FOR SBCS DATA
      WITH DEFAULT NULL,
    MSG_CRE_DT_TM       TIMESTAMP (6)
      WITH DEFAULT NULL,
    BK_TO_CSTMR_STMT    XML
      (XMLSCHEMA ID SYSXSR.CAMT_053_001_02)
       NOT NULL)
  IN BKSTDB01.BKSTTS01
PARTITION BY RANGE (MSG_CRE_DT_TM)
   (PARTITION 1  ENDING AT ('2009-12-31T23:59:59.99999'),
    PARTITION 2  ENDING AT ('2010-12-31T23:59:59.99999'),
    PARTITION 3  ENDING AT ('2011-12-31T23:59:59.99999')) #
```

Now, assume that the average size of a BankToCustomerStatement is 4 KB. The base table is small with a maximum row length of 82 bytes.

If we want to store 10,000 bank statements each year, this yields the following values (in round numbers):

- For the base table space:

  `10,000 * 82 bytes = 800 KB`

- For the XML table space:

  `10,000 * 4 KB = 40 MB`

Therefore, if we simply stay with the (default) DSSIZE of 4 GB and page size of 4 KB, we will have sufficient space for the data.

However, what if instead of 10,000 bank statements, we have 2 million each year? In this case, the approximate estimate becomes, as follows:

- For the base table space:

  `2,000,000 * 82 bytes = 160 MB`

- For the XML table space:

  `2,000,000 * 4 KB = 7,8 GB`

Therefore, even if plenty of space is available for the base table row in each partition that uses DSSIZE 4G, the limit of the XML table space partition is reached long before the 2 million rows and this setting imposes a limit on the base table also. An attempt to insert a row when the XML table space partition is full can result in an SQL code -904, indicating that the XML table space is unavailable.

The solution is either to use a more granular partitioning key, or to choose a combination of DSSIZE and page size of the base table space that will result in a larger DSSIZE for the XML table space.

### 10.2.4 Compression

Using compression can significantly reduce the amount of disk space needed to store XML data.

To compress data, use the following steps:

1. Specify COMPRESS YES in your XML table space. Note the following information:
   - The COMPRESS property of implicitly created XML table space is inherited from base table.
   - To check the COMPRESS property of your table space, query the COMPRESS column of SYSIBM.SYSTABLEPART catalog table
   - You can ALTER TABLESPACE with COMPRESS clause to change the COMPRESS property
2. Run the REORG utility.

   If the XML table space has the COMPRESS YES attribute, the XML data will be compressed

With DB2 10 NFM, you can turn on compression with ALTER any time and the compression dictionary is built when you execute the following items:

- INSERT statements
- MERGE statements
- LOAD SHRLEVEL CHANGE and SHRLEVEL NONE

Additionally, when you INSERT, MERGE, or LOAD XML data, a dictionary can be built specifically for the XML table space if the amount of XML data is large enough. Then, the new inserted XML data will be compressed.

> **Note:** Real-time statistics (RTS) keeps track of the amount of data for the threshold of online compression. Apply the PTF for APAR PM22081 to correct RTS errors when LOAD RESUME is executed on partitioned table spaces.

The compression dictionary is built through these if any of the following statements is true:

- The table space or partition is defined with COMPRESS YES.
- The table space or partition has no compression dictionary built yet.
- The amount of data in the table space is large enough to build the compression dictionary.

You can also compress the XML indexes as shown in Example 10-2.

*Example 10-2 Creating an XML index with compression*

```
CREATE INDEX LEANXML_IX3
ON BK_TO_CSTMR_STMT(BK_TO_CSTMR_STMT)
GENERATE KEY USING XMLPATTERN
'declare default element namespace
  "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02"#
/Document/BkToCstmrStmt/GrpHdr/MsgId'
AS SQL VARCHAR(35)
COMPRESS YES#
```

## 10.2.5 Registration of schemas

An XML schema consists of a set of XML schema documents. You must register the XML schema to DB2 XSR before using it. You can register an XML schema in any of the following ways:

- Call the following DB2-supplied stored procedures from a DB2 application program:
  - SYSPROC.XSR_REGISTER: Begins registration of an XML schema.
  - SYSPROC.XSR_ADDSCHEMADOC: Adds additional XML schema documents to an XML schema.
  - SYSPROC.XSR_COMPLETE: Completes the registration of an XML schema.
- Invoke the following JDBC method from a Java application program

  ```
  com.ibm.db2.jcc.DB2Connection.registerDB2XmlSchema
  ```

- Invoke the following commands from the CLP:

  ```
  -REGISTER XMLSCHEMA
  -ADD XMLSCHEMA DOCUMENT
  -COMPLETE XMLSCHEMA
  ```

We show how to register XML schema by using CLP in Example 6-2 on page 91, and how to register XML schema by using a Java program in Example 7-4 on page 140. See *DB2 10 for z/OS pureXML Guide*, SC19-2981 for more information.

As Chapter 5, “Validating XML data” on page 75 shows, the XML schema repository offers a wide range of choices for specifying XML schemas when validating XML documents against a schema, both for automatic and explicit validation.

You have the option of specifying a schema using any of the following items:

- Schema name
- URI and location hint
- Namespace

Although being able to choose the option you prefer is certainly convenient, a good approach is to decide and document what method you want to use in your company. This approach can help ensure that you always have clarity about which XML schema or schemas you are working with, without having to go through the elaborate rule-set for schema selection.

In addition, a worthwhile plan might be to define a naming standard for the XML schemas, taking into account that XML schemas can evolve over time and require several versions available in the XSR at a given time.

Neither of these are technical tasks, but they often lie within the responsibilities of the DBA and may be good to plan for when starting a new XML project.

## 10.2.6 Creation of XML indexes

You can create an index on an XML column for efficient evaluation of Xpath expressions to improve performance of queries on XML documents. In contrast to simple relational indexes where index keys are composed of one or more table columns that you specify, an XML index uses a particular Xpath expression to index paths and values in XML documents that are stored in a single XML column.

Be sure to specify a data type for every XML index. XML indexes support the data types VARCHAR, DECFLOAT, DATE, and TIMESTAMP. Example 10-3 shows how to create an XML index.

*Example 10-3 Create an XML index*

```
CREATE INDEX IXMLNTRY
ON BK_TO_CSTMR_STMT(BK_TO_CSTMR_STMT)
GENERATE KEY USING XMLPATTERN
'declare default element namespace
  "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
  /Document/BkToCstmrStmt/Stmt/Ntry/BookgDt/DtTm'
AS SQL TIMESTAMP
```

For XML indexes performance considerations, see 11.3, “Managing access path selection with XML” on page 253.

### 10.2.7 Grants and authorizations required

When a table is created with an XML column, an XML table space, XML table, a NODEID index, and document ID index are implicitly created. The privilege set must include the following privileges:

- The USE privilege on the buffer pool and the storage group that is used by the XML objects
- If the base table space is explicitly created, CREATETS privilege on the database that contains the table (DSNDB04 if the database is implicitly created)

If you add an XML column, the privilege set requires the CREATETAB and CREATETS privileges on the database that contains the table (DSNDB04 if the database is implicitly created), and USE privilege on the buffer pool and the storage group that is used by the XML objects.

The implicitly created objects are owned by the owner of the base table.

## 10.3 Housekeeping

For XML objects maintenance, you can use IBM DB2 for z/OS utilities. The utilities handle XML objects similar to the way that they handle LOB objects, as follows:

- CHECK DATA

  In addition to normal checking, the CHECK DATA utility also checks XML relationships, the integrity of XML documents, and system-generated indexes that are associated with XML data.

- LOAD/UNLOAD

  The input/output data can be in the textual XML format or the binary XML format.

  If you load data into an XML column that has an XML type modifier, the LOAD utility validates the input data according to the XML schema that is specified in the XML type modifier.

  By using spanned records, you are able to unload and load XML documents with a file size in excess of 32 KB with good performance. The crossloader capability of the LOAD utility does not support the XML data type.

- REORG TABLESPACE

  Use the REORG TABLESPACE utility to reorganize XML objects.

  When you run REORG on an XML table space that supports XML versions, REORG discards rows for versions of an XML document that are no longer needed.

- RUNSTATS

  Use the RUNSTATS utility to gather statistics for XML objects.

  In the physical implementation, an XML document may take up more than one row in the XML table space. Therefore, for the real-time statistics of XML table spaces in SYSIBM.SYSTABLESPACESTATS, the number of rows is reported, not the number of XML documents.

For more information about utility support, see Chapter 9, “Utilities with XML” on page 183.

# 10.4 Backup and recovery

As with a LOB column, an XML column holds only a descriptor of the column. The data is stored separately. Backup and recovery of XML objects are similar to backup and recovery of LOBs. A base table space must be kept consistent with its associated LOB or XML table spaces with respect to point-in-time recovery, so for backup and recovery, you group all related objects together.

Use the REPORT utility with the TABLESPACESET option to identify related objects, which may include objects related by referential integrity (RI) or auxiliary relationships to one or more XML and LOB table spaces. An example can be found at 9.14, "REPORT" on page 218.

You may also use LISTDEF with XML/LOB and RI option to include related objects as a list. See 9.6, "LISTDEF" on page 195 for more detail.

# 10.5 Diagnostics

When dealing with errors in XML table spaces and related objects, most of the problems are the same as with any other table spaces and the usual techniques can be applied to diagnose and solve the problems.

The following errors are specific to XML:

- Corrupted XML  documents. There might be missing rows in a document (which can be made up of more than one row) or structural defects to the nodes.
- Inconsistencies might exist between XML table space and NODEID index. An index entry might exist but no corresponding XML document, and vice versa.
- Inconsistencies might exists between base table space and NODEID index. A reference might exist in the base table but no corresponding entry in the NODEID index, and vice versa.
- XML documents have not been validated against any schema in an XML type modifier.
- One or more XML documents are not valid according to any of the schemas in the XML type modifier.

## 10.5.1 Identification of XML related objects

The following ways can determine the objects that are related to the XML column:

- Run the REPORT TABLESPACE utility to identify all objects that belong to the base table space set and XML table space set. An example is shown in Example 9-20 on page 218.
- Query the DB2 catalog to obtain information of all the related objects. Various example queries are shown in 4.4, "Catalog queries to gather information" on page 65.
- Use LISTDEF with keyword ALL or XML in conjunction with the utilities to include base and XML or only XML objects. For more information, see 9.6, "LISTDEF" on page 195.

## 10.5.2 Investigating XML specific errors

In problem analysis, one of the first actions is often to display the table and index spaces in question to determine whether the XML or base table is in a restricted state.

The output of a display command is shown in Example 10-4. It shows that all spaces are in read and write (RW) status except the XML table space, which is in the restricted state CHKP (check pending). The XML table space is easily identifiable with the type of XS.

*Example 10-4 Display database command shows XML table space in AUXW*

```
DSNT360I  -DB0B ***********************************
DSNT361I  -DB0B *  DISPLAY DATABASE SUMMARY
                *    GLOBAL
DSNT360I  -DB0B ***********************************
DSNT362I  -DB0B     DATABASE = DSN00155  STATUS = RW
                   DBD LENGTH = 4028
DSNT397I  -DB0B
NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
BKRTORCS TS    0001 RW
BKRTORCS TS         RW
XBKR0000 XS    0001 CHKP
XBKR0000 XS         CHKP
IRDOCIDB IX   L0001 RW
IRDOCIDB IX    L*   RW
IRNODEID IX   L0001 RW
IRNODEID IX    L*   RW
IXMLNTRY IX   L0001 RW
IXMLNTRY IX    L*   RW
******* DISPLAY OF DATABASE DSN00155 ENDED      **********************
DSN9022I  -DB0B DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

Only three restricted states are related specifically to XML, and they might appear in other contexts too with another meaning. These states are listed in Table 10-4.

*Table 10-4 Restricted states related to XML*

| Restricted state | Cause | Comment |
|---|---|---|
| AUXW on base table | Invalidated XML column as a result of running:<br>CHECK DATA AUXERROR INVALIDATE | Can also be related to a LOB column of the table. |
| ACHKP on base table | Invalid XML column detected when running:<br>CHECK DATA AUXERROR REPORT | Can also be related to a LOB column of the table. |
| CHKP on XML table | Validation of documents have not been made, although required because of change in XML type modifier. | N/A |

Run the CHECK DATA utility with the INCLUDE XML TABLESPACES keyword to determine the exact cause of any of these states, and what further action to take. For a comprehensive description of the XML-related capabilities of the CHECK DATA utility, see 9.1, “CHECK DATA” on page 184.

Run the CHECK INDEX utility if you suspect errors in any of the indexes, such as missing or extraneous entries. The task of running CHECK INDEX is applicable to the DOCID index on the base table, and the NODEID index on the XML table, and also any XML value indexes that you have created yourself. For more information about the CHECK INDEX utility, see 9.2, “CHECK INDEX” on page 189.

Run the REPAIR utility with the LOCATE KEY and LOCATE RID keywords to determine whether certain rows and index entries exist:

- LOCATE KEY can be used to locate a row in a base table by specifying the DOCID key in the DOCID index.
- LOCATE RID can be used to locate a row in the XML table space using the RID of the row.

Example 10-5 shows how to use the utility to locate a row in base table space BKRTORCS with `DocID =2`, and a row in the XML table space with `RID X'00000201'`.

*Example 10-5 REPAIR LOCATE control statements for diagnosing XML inconsistencies*

```
REPAIR LOCATE TABLESPACE DSN00155.BKRTORCS
   KEY(2) INDEX XMLR2.I_DOCIDBK_TO_CSTMR_STMT DUMP
REPAIR LOCATE TABLESPACE DSN00155.XBKR0000
   RID X'00000201' DUMP
```

**Note:** Apply the currently open APAR PM26592 to allow for use of BIGINT data type in REPAIR LOCATE KEY.

### 10.5.3 Correcting XML data

Depending on the identified type of error or inconsistency, the ways of correcting the data differ.

If the errors were identified by the CHECK DATA utility, you can use the CHECK DATA utility to correct the problems.

- Run CHECK DATA with SHRLEVEL REFERENCE and XMLERROR INVALIDATE, which causes the utility to perform the following tasks:
  - Delete invalid XML documents and move them to exception tables
  - Invalidate XML entries in the base table
- Run CHECK DATA with SHRLEVEL CHANGE and XMLERROR INVALIDATE, which causes the utility to generate control statements for you to execute:
  - REPAIR LOCATE DOCID DELETE statements for deleting orphan rows
  - REPAIR LOCATE RID REPLACE statements for invalidating entries in the base table
  - REBUILD INDEX for the NODEID index if the index is in error.

For details about the CHECK DATA utility and the options for correcting the data, see 9.1, “CHECK DATA” on page 184.

**Notes:**

Apply PTF UK62510 for APAR PM24947 for the CHECK DATA SHRLEVEL CHANGE utility to generate REPAIR statements.

APAR PM21834 (currently open) provides various usability fixes for DB2 utilities including CHECK DATA.

If the errors were identified by the CHECK INDEX utility, possibly in conjunction with REPAIR LOCATE, the appropriate course of action is listed in Table 10-5.

*Table 10-5 Corrective action after running CHECK INDEX*

| Problem | Solution |
|---|---|
| Error in DocID index | 1. Ensure that the table space is at the correct level.<br>2. Rebuild the index. |
| Mismatch between NODEID index or user-defined XML index and XML table space, and index is correct. | Use REPAIR LOCATE RID DELETE to remove orphan row. |
| Mismatch between NODEID index or user-defined XML index and XML table space, and XML table space is correct | Rebuild the index. |

Finally, you might need to reset the restrictive status by running CHECK DATA SHRLEVEL REFERENCE, or by using the REPAIR utility as shown in Example 10-6.

*Example 10-6 Using REPAIR utility to clear ACHKP status on table space*

```
REPAIR OBJECT
SET TABLESPACE DSN00155.BKRTORCS NOAUXCHKP
```

# 11

# Performance considerations

In this chapter, we provide a checklist of the major (additional) performance considerations when you deploy an application that uses pureXML.

This chapter contains the following checklist information:

- Choice of relational or XML storage
- XML schema validation
- Managing access path selection with XML
- Encourage use of native SQL DB2 routines
- External language programming
- DBA considerations
- SQL/XML coding techniques

© Copyright IBM Corp. 2011. All rights reserved.

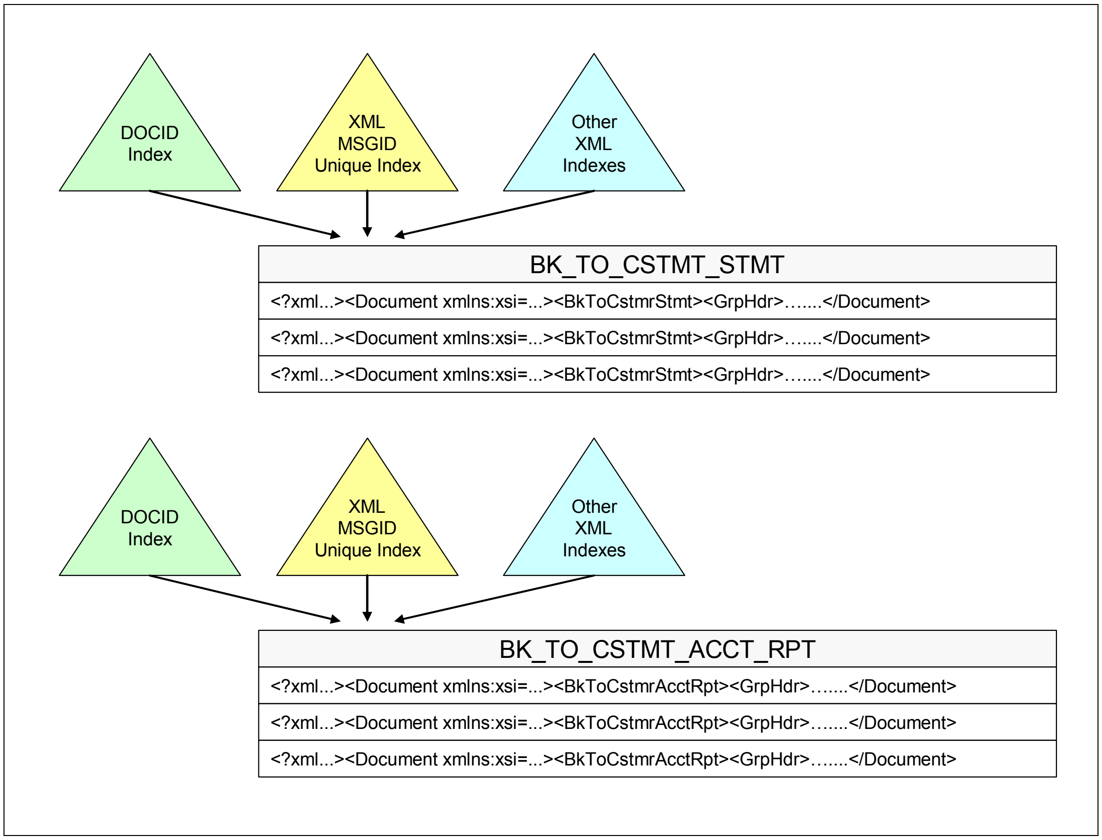
## 11.1 Choice of relational or XML storage

The first consideration is whether native XML is the right choice of storage model for persisting the data in your new application. DB2 offers three choices:

- Relational-only storage (which we do not consider a viable solution any more
- XML only storage
- Hybrid storage

### 11.1.1 XML only storage

The option to store data by using tables that contain only columns with the XML data type is a practical option. The enforcement of XML schemas, combined with the ability to create XML indexes to enforce uniqueness and XML indexes for performance, provides the ability to implement an XML database with integrity and performance. Figure 11-1 illustrates two ISO 20022 message types, covering the area of payment notifications, implemented as an XML-only database in DB2.



| BK_TO_CSTMT_STMT |
|---|
| <?xml...><Document xmlns:xsi=...><BkToCstmrStmt><GrpHdr>.......</Document> |
| <?xml...><Document xmlns:xsi=...><BkToCstmrStmt><GrpHdr>.......</Document> |
| <?xml...><Document xmlns:xsi=...><BkToCstmrStmt><GrpHdr>.......</Document> |

| BK_TO_CSTMT_ACCT_RPT |
|---|
| <?xml...><Document xmlns:xsi=...><BkToCstmrAcctRpt><GrpHdr>.......</Document> |
| <?xml...><Document xmlns:xsi=...><BkToCstmrAcctRpt><GrpHdr>.......</Document> |
| <?xml...><Document xmlns:xsi=...><BkToCstmrAcctRpt><GrpHdr>.......</Document> |

*Figure 11-1 XML-only database design*

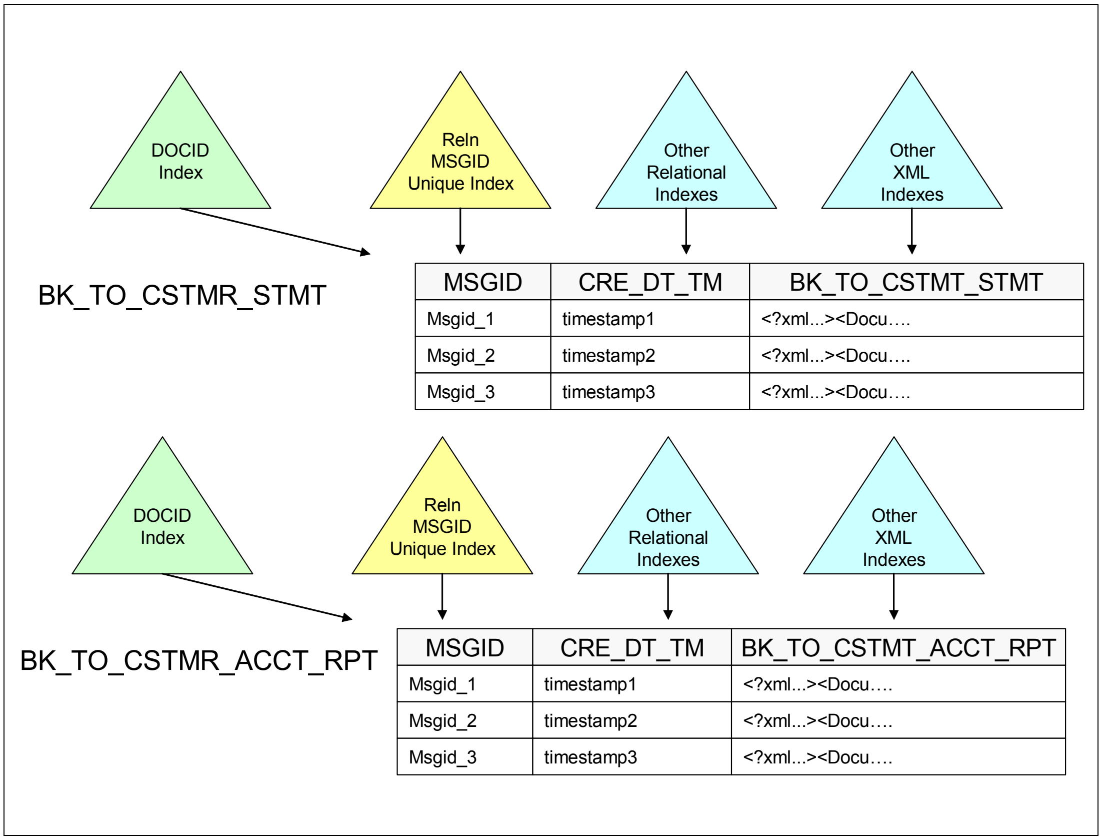
An XML-only design has the following potential drawbacks for traditional DB2 users:

- You cannot define referential integrity constraints between tables if it involves XML columns, or XML expressions against the contents of those XML columns.
- Development and DBA teams might be forced to “think” in terms of XML and XPath expressions for every database interaction that they perform. This approach is a large conceptual shift from traditional relational and XML thinking.

XML-only storage is a radical change for many DB2 users, which might stretch existing skills beyond their comfort level. For most DB2 users, a kind of hybrid storage model is best, because it can minimize the learning curve for DBAs and developers who are established relational users, without diminishing the ability to manage XML documents with pureXML.

## 11.1.2 Hybrid storage

The ability of DB2 to support a hybrid storage model, as illustrated in Figure 11-2, provides the best solution to DB2 users. The XML documents can be preserved in their original state (which may be needed for compliance reasons in certain cases) and can be accessed at any level of detail by using SQL/XML. Optionally, those data elements that benefit from being stored in relational format can be stripped out and stored in relational columns. However, do not strip XML data elements out, into relational storage, unless there is a clear benefit in doing so.



| MSGID | CRE_DT_TM | BK_TO_CSTMT_STMT |
|---|---|---|
| Msgid_1 | timestamp1 | <?xml...><Docu…. |
| Msgid_2 | timestamp2 | <?xml...><Docu…. |
| Msgid_3 | timestamp3 | <?xml...><Docu…. |

| MSGID | CRE_DT_TM | BK_TO_CSTMT_ACCT_RPT |
|---|---|---|
| Msgid_1 | timestamp1 | <?xml...><Docu…. |
| Msgid_2 | timestamp2 | <?xml...><Docu…. |
| Msgid_3 | timestamp3 | <?xml...><Docu…. |

*Figure 11-2 Hybrid storage model*

The programming examples in this book (SQL procedures, COBOL and Java) all show how using the SQL/XML language can be easy to strip out data elements of interest, and store them as relational columns.

Although this approach is not necessary to achieve a high performance database, it can be desirable for many other reasons:

- Referential Integrity can be defined on the indexed relational columns.
- The database is easier to work with, because it can be represented easily in the familiar representation of entity-relationship diagrams, which is the conceptual model upon which traditional relational systems have been designed for years.

- The contents of the XML documents can be accessed with ease, using the power of the SQL/XML language.
- Developers do not have to be highly skilled in XML expressions, particularly if the database is implemented with appropriate use of stored procedures and user-defined functions (to encapsulate XML functions in callable SQL routines). With this approach, developers can work effectively with the database by using standard SQL most of the time.
- Data structures that are naturally best supported in an XML structure can be stored unaltered in DB2 in their optimal physical representation, and can be accessed efficiently using XML expressions in SQL/XML.
- The inherent strengths of the XML structures can still be fully utilized with SQL/XML, regardless of the fact that we have also stored some of the data elements in relational columns to make some development and DBA tasks easier.
- Relational and XML data can be joined and processed together with ease and good performance. New applications can be written to fully utilize pureXML, while being able to integrate seamlessly with traditional relational structures.
- Data that is stable in terms of its structure (low variability) and accessed often can be a good fit for extraction into relational columns. More variable parts of the data are better kept in XML.

### Dedicated use of XML columns

Another database design consideration is whether having one XML column to contain XML documents of differing XML schemas is better than having multiple columns that each contain documents of one specific XML schema.

Either approach is a valid trade-off between database management complexity and performance.

If you have multiple XML columns to store XML documents of separate XML schemas, you can create many more database objects (table spaces, data sets, DOCID indexes, NODEID indexes, and so on). The logical data model is easier to understand if separate XML document types are stored in their columns. However, the physical data model can have a larger number of objects to track.

XML Indexes and performance might suffer if you combine multiple XML document types into a single DB2 column. If you create an XML index on an XML column with multiple XML schemas, the costs of index creation can increase, even if many of the documents do not contain matching nodes. Additionally, if you are not careful about XPath expressions and namespaces, you might receive unexpected results if you get an index hit against an XML document type that you were not targeting in your application.

As a general rule, a good approach usually is to store XML documents in DB2 columns that are dedicated to documents of that particular XML schema.

## 11.1.3 Natural fit for XML storage

Certain data is an obvious natural fit for pureXML storage. The ISO 20022 standard for banking messages that are used in the application scenarios of this book is a good example. These messages are defined by ISO as a standard for messaging for European banking. Storing these messages in exactly the way that they were transmitted is important, because they are a record of financial transactions. With DB2 pureXML, we can store these messages in their original format, and also query and interrogate them by using the power of SQL/XML, and the efficiency of pureXML.

There are a large range of other industry standard models for XML storage and XML messaging, particularly in financial services, but also in other industries. We might want to store data in a persistent database for both query and transactional purposes. DB2 pureXML provides a good foundation for such systems.

Many data structures for in-house systems might be best implemented by using XML. Consider the example of insurance quotations. The range of data that is collected for a car insurance proposal can vary dramatically depending on the vehicle, the applicants, the usage of the vehicle, the claims history, and so on. Do we really want to design a relational database with 50 or so tables to store millions of Internet quotes, when they are sparsely populated and only 1% of them will be taken forward into a purchased policy? Surely this is an example of an in-house data structure that should implemented in XML. Price comparison sites must surely use XML documents and web services to communicate between separate insurers. With pureXML, these documents can be stored for efficient retrieval, and indexed and queried in their native format for patterns in client behavior.

The following data characteristics tend to make XML a good choice for data storage:

- Hierarchical data

  The XML data model is naturally hierarchical, which means that it tends to be a good fit for data that is naturally hierarchical.

- Semi-structured data

  XML schemas can be rigid or loose as appropriate. They can have rigid constraints for certain XML nodes, and looser constraints for other XML nodes as appropriate, to accommodate the appropriate level of structure that is required.

- Document/narrative data

  XML is always accused of being verbose.

- Many types of schemas

  The ISO 20022 standard is an example. It contains hundreds of schemas. The development cost of building relational database schemas for all these message structures would be astronomical. With pureXML, the number of tables and columns can be very small, because the data structures are managed in the XML schemas, with little DBA impact.

- Large schemas (with sparsely populated attributes and elements)

  The ISO 20022 schemas are very large to provide a generic schema that can cater to a very wide range of payments scenarios. The average personal banking customer uses only a subset of the facilities in that schema. No additional processing or storage costs are incurred for leaving the majority of the XML nodes empty of data. XML indexes contain only entries for those XML patterns where an indexed element or attribute is found, thus minimizing the space required for XML indexes.

- Quickly evolving schema

  If a relational database schema changes, months can be spent for the development, testing, and QA processes to introduce a new release of the application. A major part of that release time is spent by database change management. This part would be substantially reduced if all you had to do was add a new schema document to the existing XSR schema, to define a new release of the XML schema. Furthermore, database schema changes do not necessarily require data migration; the application can decide whether an XML document which is compliant with the old schema release must be modified to accommodate the new schema release.

- Forms-based applications

  If the data from a forms-based application is created in XML, there is not reason to convert it to relational for storage.

- Data with nulls and multiple values

  XML storage is a way for DB2 to support multi-value data storage.

- Existing industry standard schemas for XML

  The ISO 20022 examples in this book show how you can easily incorporate an existing XML standard into DB2, and use it with a minimal amount of database administration work. There are many other industry-related XML standards, some of which are listed in Chapter 1, “Introduction” on page 1.

If the data for your application fits some or all of these characteristics, then pureXML can be the best physical storage option. If you choose pureXML storage, consider carefully the optimal hybrid storage design, which can help make development tasks easy, and allows the strengths of the XML model to be fully utilized.

## 11.2 XML schema validation

Having chosen pureXML for your storage model, and the appropriate degree of hybrid storage, the next questions are whether and when to perform schema validation. The primary considerations are as follows:

- Data integrity is paramount. XML schemas are the method by which the integrity rules of XML data are defined, and XML schema validation is how they are enforced.
- Schema validation is CPU-intensive, so we do not want to validate XML documents unnecessarily or repeatedly.
- XML schema validation is a candidate for zIIP and zAAP redirection, which means that XML schema validation need not increase your general purpose CPU resource consumption.

Consider the following guidance:

- Do not take chances with data integrity. If you are not sure that an XML document is valid (perhaps if it was received from an external source, which is a common scenario), be sure to validate it before you commit it to your database.
- If you know that an XML document has been validated against its schema (perhaps because it was validated in WebSphere Message Broker before being stored in DB2), do not revalidate simply for the purpose of revalidation.
- In scenarios where you always want to perform schema validation, ensure that the DB2 table is defined with an XML type modifier, so that XML validation cannot be bypassed.
- In scenarios where the some, but not all, of the XML documents have been validated before they are presented to DB2, do not define the DB2 table with an XML type modifier. In this case, make use of the DSN_XMLVALIDATE function to perform validation only where necessary.

- You may optionally control whether certain users or programs need to perform XML validation by restricting access to the base DB2 table, and encapsulating write access (with or without XML schema validation calls) to the DB2 table within stored procedures or user-defined functions. Access to these routines and functions can be granted to controlled groups as appropriate.
- You may implement automated XML validation in test and quality assurance environments so that you can flush out XML data quality issues before an application is deployed to the production environment. The amount of XML validation that is enforced in a production environment with higher transaction volumes can be set to a lower level if the XML documents in a production environment are known to be valid.

> **Note:** The XML validation process (whether invoked manually through DSN_XMLVALIDATE, or automatically through DDL table modifier) always requires a string data type as input. If you invoke XML schema validation against an XML data type, the validation first converts the XML document to a string value, and then performs XML schema validation. Validation using binary XML is not yet supported.

Be sure to also monitor zIIP and zAAP use, because XML processing is 100% eligible. For more information about zIIP and zAAP monitoring, go to the following address:

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101227

## 11.3 Managing access path selection with XML

XML indexes can be used instead of relational indexes, or in addition to relational indexes. The critical consideration is that DB2 indexes (of one sort or the other) are used to eliminate the retrieval and searching of a large number of XML documents. If scanning a relational table space is bad for performance, then the prospect of scanning and processing an XML table space can be far worse.

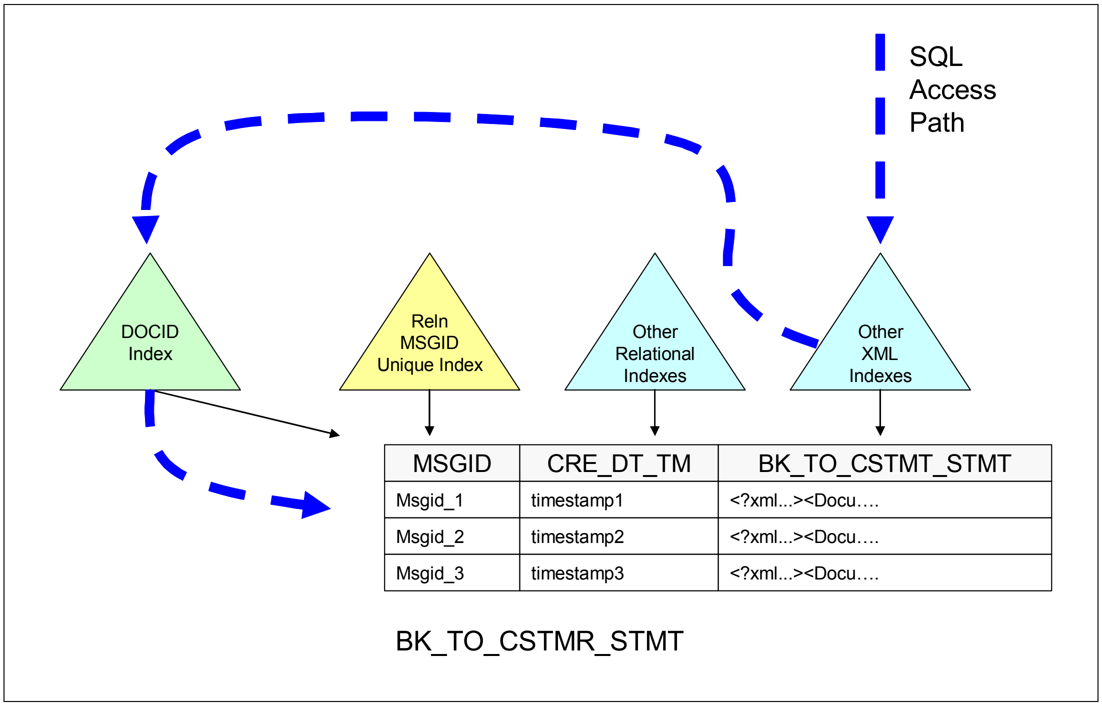
The principles of designing access paths are no different from relational access path selection, as follows:

- XML indexes are B-tree structures, like relational indexes.
- The DB2 Optimizer considers using both XML and relational indexes, based on their attractiveness.
- The attractiveness of XML indexes is based on largely the same measures as relational indexes (cardinality, filter factors, and others that you find in the DB2 catalog).

The best way of approaching the subject of XML indexes for access path selection is to be aware of the differences between XML index usage and relational index usage, and add this knowledge to traditional quality assurance processes that you already use for access path selection.

### 11.3.1 Differences between XML and relational indexes

The purpose of an XML index is to retrieve a number of DOCID values, which can be used to perform filtering of qualifying rows in a table, as Figure 11-3 on page 254 shows. Subsequent table access can be performed through the index on the hidden DB2_GENERATED_DOCID_FOR_XML column.



| MSGID | CRE_DT_TM | BK_TO_CSTMT_STMT |
|---|---|---|
| Msgid_1 | timestamp1 | <?xml...><Docu…. |
| Msgid_2 | timestamp2 | <?xml...><Docu…. |
| Msgid_3 | timestamp3 | <?xml...><Docu…. |

*Figure 11-3 Physical access path using an XML Index.*

XML indexes are created on an XML pattern (examples are listed in “XML Index patterns” on page 255). The result of that XML pattern yields a result, which can be variable in nature, depending on the content of the XML document and the constraints of an XML schema.

The differences between relational and XML indexes are listed in Table 11-1.

*Table 11-1 Differences between relational and XML indexes*

| **Relational indexes** | **XML indexes** |
|---|---|
| May be defined on one or more relational columns. | Can be defined only on one XML element or attribute (using an XML pattern expression). |
| Always have one index entry for every row in a table. XML indexes are much less prescriptive. | Are based on an XML pattern. An XML pattern may occur any number of times in an XML document. Therefore, XML indexes may contain 0, 1, or many entries for each row in the table. |
| Are always based on the data types of the column(s) that they are defined on. The data types found at the locations of an XML pattern may be many and varied unless an appropriate XML schema is enforced. | Are defined based on a mapping to a particular data type, but whether or not the data type that is found at that location can be cast to that data type for an index match to be achieved depends on the degree to which the XML schema constrains the data contents. |
| Can be used to support table clustering. | May not be used for table clustering support. |

Now that you understand the nature of XML indexes and how they differ from relational indexes, you can use that information when considering how to design XML indexes.

## 11.3.2 XML index design

The differences between XML indexes and relational indexes leads to a set of XML index design considerations, which are described in this section.

### XML Index patterns

A wide range of XML patterns can be used in XML indexes.

At the most restrictive end of the spectrum are *lean* indexes. The index in Example 11-1 is targeted at a data element in a specific XPath location, which is cast to a specific relational data type. In this example, an index is built and is based on mapping the data elements at `XPath /Document/BkToCstmrStmt/GrpHdr/MsgId` and casting whatever is found there to `Varchar(35)`.

> **Note:** XMLPATTERNS must define the appropriate namespace if the Documents contain a namespace declarations.

*Example 11-1 A lean XML index*

```
Create index LEAN_XMLIX
   ON BK_TO_CSTMR_STMT(BK_TO_CSTMR_STMT)
   Generate Key using XMLPATTERN 'declare default element namespace
   "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      /Document/BkToCstmrStmt/GrpHdr/MsgId'
as SQL VARCHAR(35)  ;
```

At the least restrictive end of the spectrum are *heavy* indexes. The index in Example 11-2 is targeted at any occurrence of a data element, MsgId, within the entire document.

*Example 11-2 A heavy XML index*

```
Create index HEAVY_XMLIX
   ON BK_TO_CSTMR_STMT(BK_TO_CSTMR_STMT)
   Generate Key using XMLPATTERN 'declare default element namespace
   "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      //MsgId'
as SQL VARCHAR(35)  ;
```

Choosing lean or heavy XML indexes is a physical design trade-off. You want to maximize the filter factor of every index, but you might be prepared to compromise the filter factor if you can get away with a smaller number of indexes.

When you create your first XML index, you will want to get confirmation about whether it is doing the task that you want it to do. The obvious way to do this is to explain a query that should benefit from the index, and see if the index is selected by the optimizer.

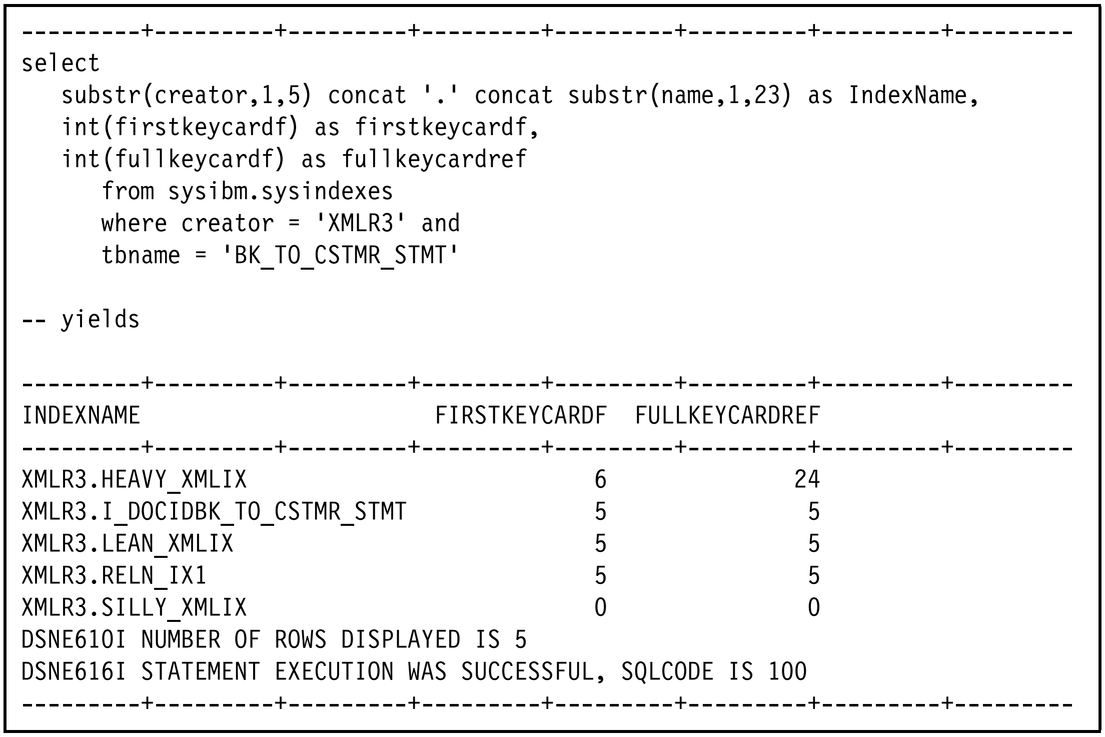
Consider the “silly” XML index in Example 11-3 on page 256. This index was created with an XMLPATTERN that did not actually match any nodes in the XML document, because of a typographical error in the xpath expression.

*Example 11-3 A “silly” XML index*

```
Create index SILLY_XMLIX
   ON BK_TO_CSTMR_STMT(BK_TO_CSTMR_STMT)
   Generate Key using XMLPATTERN 'declare default element namespace
   "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      /Document/BkToCstmrStmt/MsgId'
as SQL VARCHAR(35)  ;
```

If you find that the index is not selected, and you are unable to determine why the optimizer is not choosing the index, take a moment to check the contents of SYSIBM.SYSINDEXES. The table snapshot in Figure 11-4 shows a subset of the catalog statistics for the lean and the heavy indexes that were created in Example 11-1 on page 255 and Example 11-2 on page 255. It also shows the catalog statistics of another XML index, which is based on an XML pattern with a typographic error. See Figure 11-4. Note the following information:

- The table has five rows, each with an XML document.
- The lean XML index has a cardinality of 5, because it is based on an XPath expression that identifies the MsgId data element, which happens to be unique.
- The heavy XML index has a cardinality of 24, because the MsgId data element is repeated multiple times in a typical Bk_To_Cstmr_Stmt message.
- The “silly” index has a cardinality of 0!



```
---------+---------+---------+---------+---------+---------+---------+---------
select
   substr(creator,1,5) concat '.' concat substr(name,1,23) as IndexName,
   int(firstkeycardf) as firstkeycardf,
   int(fullkeycardf) as fullkeycardref
      from sysibm.sysindexes
      where creator = 'XMLR3' and
      tbname = 'BK_TO_CSTMR_STMT'

-- yields

---------+---------+---------+---------+---------+---------+---------+---------
INDEXNAME                      FIRSTKEYCARDF  FULLKEYCARDREF
---------+---------+---------+---------+---------+---------+---------+---------
XMLR3.HEAVY_XMLIX                          6              24
XMLR3.I_DOCIDBK_TO_CSTMR_STMT              5               5
XMLR3.LEAN_XMLIX                           5               5
XMLR3.RELN_IX1                             5               5
XMLR3.SILLY_XMLIX                          0               0
DSNE610I NUMBER OF ROWS DISPLAYED IS 5
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+---------
```

*Figure 11-4 Catalog query to SYSIBM.SYSINDEXES*

The fact that FIRSTKEYCARDF and FULLKEYCARDF are zero is the warning alarm that tells you that the index did not point to any matches in any of the XML documents in the table. When this situation happens, you can deduce that you have an error in your XML pattern in the index definition. Check the XML pattern, correct it, re-create the index, run RUNSTATS and review the cardinality statistics until you get a sensible number.

This way is a useful way to check that the index you have created is based on an XML pattern that finds hits in the data. This example illustrates one of the differences between relational indexes and XML indexes. Although creating a relational index that does not have a pointer to every row in the table is not possible, the possibility does exist for having a perfectly valid XML index that points to zero rows in the table.

**Learning curve:** There is a learning curve for the SQL programmer to become comfortable with writing XML expressions such as XMLTABLE, XMLQUERY, and XMLEXISTS. If you code an invalid SQL/XML statement, you receive an SQL error (such as `SQLCODE -104`), which usually contains helpful guidance about what is wrong with your statement. Further frustration is when you code an SQL/XML statement that is valid, but it returns no rows. The two most common causes of getting no rows are XPath expressions containing a typographic error, and incorrect namespace declarations. These two reasons are also the first causes to consider when an XML index matches nothing in the table.

### XML index maintenance cost

XML indexes are more expensive for insert than relational indexes because DB2 has more work to perform. Consider inserting a row into a DB2 table with an integer index and an XML index. Ignoring factors such as page splits, the maintenance of the relational index is simply a case of inserting a new entry into the appropriate index leaf page. However, the maintenance of the XML index requires that the XML document be scanned for zero, one or many matches to the XMLPATTERN of the index, and also the insert of a new value into the index leaf page. The maintenance of an XML index is similar to the maintenance of an index-on-expression (with the XPath XMLPATTERN being the *expression*).

XML indexes are also more expensive for selection than relational indexes because two indexes must be used:

- The NODEID index on the XML table space must be used to find matches for the XML pattern that is being searched on.
- The DOCID index must be used next, to retrieve RIDs to perform table access.

Therefore, use care when choosing XML indexes, because they can increase the path length of insert operations more than relational indexes. Do *not* interpret this statement to mean that all searchable data fields must be stripped from XML documents and placed in indexed DB2 columns. The fact that an XML index adds some extra path length must be balanced against the flexibility of being able to index directly into any part of a large XML document without having to maintain a copy of that data element in a separate relational column.

### XML index eligibility

Be aware of several index eligibility constraints when you design XML indexes. When you consider index eligibility, a useful aspect to remember is that the purpose of an XML index is to return a small (hopefully) number of DOCID values that are used for table access.

XMLEXISTS functions are eligible for index access, if the XMLEXISTS predicate is supported by the XML pattern of the XML index. Example 11-4 on page 258 shows an SQL/XML query with an XMLEXISTS predicate that can be supported by either the lean or the heavy XML indexes. In this case, the lean XML index is chosen, as proven by the results of an **`explain`** request.

*Example 11-4 Explain for XMLEXISTS*

```
explain plan set queryno = 99 for
   select c.msg_id, c.msg_cre_dt_tm
      from BK_TO_CSTMR_STMT c
      where    xmlexists('declare default element namespace
         "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
         $i/Document/BkToCstmrStmt/GrpHdr[MsgId="AAAASESS-FP-STAT002"]'
      passing c.BK_TO_CSTMR_STMT as "i");

select
   planno, creator, tname, accesstype,
   matchcols, accesscreator, accessname, table_type
      from XMLR3.PLAN_TABLE where queryno = 99;

PLANNO        = 1
CREATOR       = XMLR3
TNAME         = BK_TO_CSTMR_STMT
ACCESSTYPE    = DX
MATCHCOLS     = 1
ACCESSCREATOR = XMLR3
ACCESSNAME    = LEAN_XMLIX
TABLE_TYPE    = T
```

XMLTABLE without a filtering predicate is not generally eligible for XML index access. The reason is because the result of an XMLTABLE operation is simply a table without any filtering of DOCIDs, unless a predicate is included with the XMLTABLE function. Example 11-5 shows the use of the XMLTABLE function in conjunction with an XMLEXISTS predicate. The access path is to use the lean XML index to retrieve the DOCIDs, and then to use the table function X (which is the name that is assigned to the result of the XMLTABLE function) to retrieve the data values.

*Example 11-5 Explain for XMLTABLE with XMLEXISTS*

```
EXPLAIN PLAN SET QUERYNO = 88 FOR
   SELECT X.MSG_ID, X.CRE_DT_TM  FROM BK_TO_CSTMR_STMT as C,
      XMLTable(XMLNAMESPACES(DEFAULT
      'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02',
      '$d/Document/BkToCstmrStmt'
         PASSING c.BK_TO_CSTMR_STMT as "d"
         COLUMNS
            "MSG_ID" VARCHAR(35) PATH './GrpHdr/MsgId/text()',
            "CRE_DT_TM" TIMESTAMP PATH './GrpHdr/CreDtTm/text()' ) AS X
   where xmlexists('declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $i/Document/BkToCstmrStmt/GrpHdr[MsgId="AAAASESS-FP-STAT002"]'
      passing c.BK_TO_CSTMR_STMT as "i");

select
   planno, creator, tname, accesstype, matchcols,
   accesscreator, accessname, table_type
      from XMLR3.PLAN_TABLE where queryno = 88;

PLANNO        = 1
CREATOR       = XMLR3
TNAME         = BK_TO_CSTMR_STMT
```

```
ACCESSTYPE    = DX
MATCHCOLS     = 1
ACCESSCREATOR = XMLR3
ACCESSNAME    = LEAN_XMLIX
TABLE_TYPE    = T

PLANNO        = 2
CREATOR       = XMLR3
TNAME         = X
ACCESSTYPE    = R
MATCHCOLS     = 0
ACCESSCREATOR =
ACCESSNAME    =
TABLE_TYPE    = F
```

Another way of getting XMLTABLE to use an XML index does not require the use of XMLEXISTS. You may specify a predicate on an XMLTABLE function, as Example 11-6 shows, and get XML index access.

*Example 11-6 Explain for XMLTABLE with an XML predicate*

```
EXPLAIN PLAN SET QUERYNO = 55 FOR
   SELECT X.MSG_ID, X.CRE_DT_TM
      FROM BK_TO_CSTMR_STMT as C,
      XMLTable(XMLNAMESPACES(DEFAULT
         'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02',
         'http://www.w3.org/2001/XMLSchema-instance' AS "xsi"),
         '$d/Document/BkToCstmrStmt[GrpHdr/MsgId="AAAASESS-FP-STAT002"]'
            PASSING c.BK_TO_CSTMR_STMT as "d"
            COLUMNS
               "MSG_ID"    VARCHAR(35)  PATH './GrpHdr/MsgId',
               "CRE_DT_TM" TIMESTAMP    PATH './GrpHdr/CreDtTm/text()' ) AS X

select
   planno, creator, tname, accesstype, matchcols,
   accesscreator, accessname, table_type
   from XMLR3.PLAN_TABLE where queryno = 55;

PLANNO        = 1
CREATOR       = XMLR3
TNAME         = BK_TO_CSTMR_STMT
ACCESSTYPE    = DX
MATCHCOLS     = 1
ACCESSCREATOR = XMLR3
ACCESSNAME    = LEAN_XMLIX
TABLE_TYPE    = T

PLANNO        = 2
CREATOR       = XMLR3
TNAME         = X
ACCESSTYPE    = R
MATCHCOLS     = 0
ACCESSCREATOR =
ACCESSNAME    =
TABLE_TYPE    = F
```

XMLQUERY is not eligible for XML index access, because the purpose of XMLQUERY is to return an XML document. XMLQUERY is never used to filter the rows in a table. Therefore, XMLQUERY in general must be used in conjunction with other relational or XML predicates that will be used for filtering, as illustrated in Example 11-7.

*Example 11-7 Explain for XMLQUERY with XMLEXISTS*

```
EXPLAIN PLAN SET QUERYNO = 77 FOR
   select xmlquery('declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $d/Document/BkToCstmrStmt/Stmt' passing BK_TO_CSTMR_STMT as "d")
         from BK_TO_CSTMR_STMT C
   where xmlexists('declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $i/Document/BkToCstmrStmt/GrpHdr[MsgId="AAAASESS-FP-STAT002"]'
      passing c.BK_TO_CSTMR_STMT as "i");

select
   planno, creator, tname, accesstype, matchcols,
   accesscreator, accessname, table_type
   from XMLR3.PLAN_TABLE where queryno = 77;

PLANNO        = 1
CREATOR       = XMLR3
TNAME         = BK_TO_CSTMR_STMT
ACCESSTYPE    = DX
MATCHCOLS     = 1
ACCESSCREATOR = XMLR3
ACCESSNAME    = LEAN_XMLIX
TABLE_TYPE    = T
```

## 11.4 Encourage use of native SQL DB2 routines

Native SQL stored procedures are probably the second most important DB2 device (after XML indexes) to ensure good performance and efficiency in your pureXML applications.

Native SQL procedures in DB2 can be an excellent environment for developing high performance applications, with or without XML. They are covered in *DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7604, which explains several of their strengths as follows:

- Multiple procedural steps can be executed within DB2, eliminating network delays that an external program would experience making each SQL call and waiting for the results.
- Stored procedures run entirely within the DB2 engine.
- Procedural statements are converted to a native representation that is stored in the DB2 catalog and directory, as it is done with other SQL statements.
- They are zIIP-eligible if they are called through DRDA with TCP/IP, type 4 Java.

The advent of pureXML enhances the attractiveness of native stored procedures for the following reasons:

- You can pass huge arrays of data to and from native stored procedures, through an XML IN or OUT data type.
- You can avoid code page translations for XML processing. A stored procedure can retrieve an XML document from a DB2 table and work with it, entirely within the DB2 engine. By contrast, programming environments such as COBOL that typically work in EBCDIC must convert the contents of the UTF-8 XML document into EBCDIC, to be able to work with the data.
- XML procedures can receive an external XML document and parse it once, and then process it with SQL/XML many times without reparsing it.

In addition to the performance benefits of native stored procedures for XML processing, considerable development productivity benefits can be gained from creating frequently used procedures that encapsulate XML processing, and can be called by a developer by using standard SQL.

Native stored procedures provide a high performance platform for the programming of SQL and SQL/XML routines, which can be called from any application environment. Any application development project that uses DB2 z/OS must evaluate which programming functions should be implemented as native stored procedures, so that they can used and shared by all applications and application environments that use DB2.

## 11.5 External language programming

The biggest performance consideration for using external programming languages is the need to perform code-page translations. Chapter 8, "Using XML with COBOL" on page 157 explains that COBOL programs are based in an EBCDIC environment, but the XML is stored in UTF-8. Therefore if you want to exchange data between COBOL and pureXML in DB2, you must perform code-page translations. That chapter also describes the options to minimize the amount of code-page translation that is performed.

The code page challenge is reduced for Java when the binary XML format is used with the IBM Data Server Driver for JDBC and SQLJ, which presents binary XML data to the application only through the XML object interfaces.

## 11.6 DBA considerations

Several database administration considerations can significantly affect the performance of pureXML applications.

### Use DB2 10 NFM universal table spaces

XML versioning allows concurrency when updating XML documents. XML versioning requires the underlying table space to be defined as universal table space in DB2 10 new-function mode.

XML versioning is the mechanism that allows concurrent read access to an XML document while another user is updating it. It works by maintaining multiple copies of the XML document in the XML auxiliary table. It depends on the columns in the XML auxiliary table that are created in DB2 10 NFM. Although querying the XML table space directly is disabled, Example 11-8 shows a catalog query and result that shows the columns that get created automatically in DB2 10 NFM.

*Example 11-8 The SYSIBM.SYSCOLUMNS contents for auxiliary XML table space*

```
select name, colno, coltype, length
   from sysibm.syscolumns
      where tbcreator = user
      and tbname = 'XBK_TO_CSTMR_STMT'
         order by colno ;

... yields

NAME         COLNO  COLTYPE  LENGTH
------------ ------ -------- ------
DOCID             1 BIGINT        8
MIN_NODEID        2 VARBIN      128
XMLDATA           3 VARBIN    15850
START_TS          4 BINARY        8
END_TS            5 BINARY        8
```

The XML table is structured as follows:

- The XML document (minus tags, which are stored in SYSIBM.SYSSTRINGS) is stored in the XMLDATA column, in one or more rows, depending on the size of the document.
- If the XML document is split into multiple rows, the minimum XML node ID is stored against each row, and is indexed by the node_id index.
- The DOCID is also stored in every row.
- The start and end time stamps for each row are stored.

The DB2 10 multiversioning table space format depends on the use of a cleanup service request block (SRB) to remove old versions of XML documents that are created by UPDATE and DELETE. It is a small overhead, but can result in some CPU activity on the 16 KB buffer pool for the data and the other buffer pool for the indexes even when XML applications are idle.

## Do not use DB2_GENERATED_DOCID_FOR_XML as a key

The DB2_GENERATED_DOCID_FOR_XML column is not explicitly hidden, and is populated with a unique sequence number, and indexed, which makes it tempting to use as a primary key.

This column is an internal object, and IBM does not guarantee to keep it unchanged in the future. Do not develop applications that depend on this column.

## Data compression

The storage of XML documents automatically receives a certain amount of compression. The XML tags are stored in SYSIBM.SYSSTRINGS and replaced with binary values. However, the contents of the data values in XML documents are not compressed by default.

The textual nature of many XML documents means that they are particularly well-suited to compression techniques, which can lead to performance gains from reduced I/O and efficient use of buffer pool. Therefore, consider compressing DB2 tables with XML, and DB2 XML indexes.

After you specify COMPRESS YES in your DDL, the table space will be compressed at REORG utility execution time or with the new online compression available with DB2 10 during insert type operations.

The DSN1COMP utility can be used to assess the compression benefit, as usual.

### Remember to use the REORG utility

You may not be allowed to view the contents of XML table spaces directly, but you are still responsible for reorganizing them.

The XML table space follows the same partitioning scheme as the base table space, but has the potential to grow much faster. Make sure that you reorganize (with REORG) the XML table spaces frequently enough to maintain performance.

The method of reorganizing an XML table space requires that you build a REORG job for the base table space, and then you add an additional REORG control statement for each of the XML table spaces. The table spaces for the BK_TO_CSTMR_STMT example table are reorganized by using the JCL in Example 11-9. You must also specify the WORKDDN keyword on the REORG for the XML table space and provide the specified temporary work file. The default is SYSUT1.

*Example 11-9 Reorganization of a table space with an XML table space*

```
//REORG1 EXEC DSNUPROC,SYSTEM=DB0B,
//             LIB='DB0BT.SDSNLOAD',
//             UID=''
//DSNUPROC.SYSPUNCH DD DSN=XMLR3.DB0B.CNTL.XMLR3DB.TSAUDIT1,
//             DISP=(MOD,CATLG),
//             SPACE=(TRK,(5,5),RLSE),
//             UNIT=SYSDA
//DSNUPROC.SYSREC DD DSN=XMLR3.DB0B.UNLD.XMLR3DB.TSAUDIT1,
//             DISP=(MOD,CATLG),
//             SPACE=(TRK,(15,5),RLSE),
//             UNIT=SYSDA
//DSNUPROC.SYSUT1 DD DSN=XMLR3.SYSUT1,
//             DISP=(MOD,DELETE,CATLG),
//             SPACE=(TRK,(5,5),RLSE),
//             UNIT=SYSDA
//DSNUPROC.SORTOUT DD DSN=XMLR3.SORTOUT,
//             DISP=(MOD,DELETE,CATLG),
//             SPACE=(TRK,(5,5),RLSE),
//             UNIT=SYSDA
//DSNUPROC.SYSIN DD *
REORG TABLESPACE XMLR3DB.TSAUDIT1
      STATISTICS TABLE(ALL)
                 INDEX(ALL)
REORG TABLESPACE XMLR3DB.XBKR0000
      STATISTICS TABLE(ALL)
                 INDEX(ALL)
                 WORKDDN(SYSUT1)
/*
```

### 16 KB buffer pool

XML table spaces are always defined in a 16 KB buffer pool. You must monitor the size of the 16 KB buffer pool, and ensure that it is appropriately sized and backed up by real storage.

#### DSNZPARM settings

XMLVALA and XMLVALS are DSNZPARM settings that control the amount of virtual storage in the XML pool, which is used as working storage for document materialization, and XPath evaluation.

As described in 10.1.3, “Sizing XMLVALA and XMLVALS” on page 235, the XMLVALA subsystem parameter specifies, in KB, an upper limit for the amount of storage that each user is to have for storing XML values:

- Acceptable values: 1 to 2,097,152 KB
- Default: 204,800 KB

The XMLVALS subsystem parameter specifies, in MB, an upper limit for the amount of storage that each system can use for storing XML values:

- Acceptable values: 1 to 51200 MB
- Default: 10240 MB

Remember that these values might need to be adjusted if your application materializes a large volume of XML data.

#### RID pool size

You might to increase the RID pool size because XML index access (particularly DOCID with logical AND or OR) also uses RID pool.

## 11.7 SQL/XML coding techniques

Coding SQL/XML is a new skill for many established DB2 programmers. With several new concepts to learn in the SQL/XML language, sometimes determining “how” to write the SQL/XML statement that you want might be difficult. You will also want to work out what programming techniques will actually yield the best performance. This section provides a collection of several common coding techniques that can yield efficient data access.

### 11.7.1 XMLTABLE to minimize database calls

The XMLTABLE function is a powerful way to minimize the number of DB2 database calls to access XML data. Although retrieving XML data elements is possible by using the XMLQUERY function multiple times, a more efficient way is to replace multiple XMLQUERY calls with a single XMLTABLE call, as shown in Example 11-10.

*Example 11-10 Multiple XMLQUERY calls replaced with a single XMLTABLE call*

```
SET V_CREDTTM = (
   select xmlcast(xmlquery('declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $d/Document/BkToCstmrStmt/GrpHdr/CreDtTm'
      passing VALIDXML as "d") as timestamp) from sysibm.sysdummy1  );

SET V_MINISTMT = (
   select xmlquery('declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $d/Document/BkToCstmrStmt/Stmt'
      passing VALIDXML as "d")  from SYSIBM.SYSDUMMY1 ) ;
```

```
-- can be replaced by

SELECT X.CRE_DT_TM, X.MINISTMT INTO V_CREDTTM, V_MINISTMT
   FROM XMLTable(XMLNAMESPACES(DEFAULT
      'urn:iso:std:iso:20022:tech:xsd:camt.053.001.02',
      '$d/Document/BkToCstmrStmt' PASSING VALIDXML as "d"
         COLUMNS
            "CRE_DT_TM" TIMESTAMP PATH './GrpHdr/CreDtTm/text()'
            "MINISTMT" XML PATH './Stmt' ) AS X ;
```

Another way of making the code more elegant and more efficient is to combine the two XMLQUERY calls into a single select statement, as shown in Example 11-11.

*Example 11-11 Single select statement combining two xmlquery expressions*

```
select
   xmlcast(xmlquery('
      declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $d/Document/BkToCstmrStmt/GrpHdr/CreDtTm'
      passing VALIDXML as "d") as timestamp)
         into  V_CREDTTM,
   xmlquery('declare default element namespace
      "urn:iso:std:iso:20022:tech:xsd:camt.053.001.02";
      $d/Document/BkToCstmrStmt/Stmt'
      passing VALIDXML as "d")
         into V_MINISTMT
from sysibm.sysdummy1 );
```

## 11.7.2 XMLEXISTS for index access

Whether you use an XML or a relational index to reduce the number of rows that need to be accessed in the base table does not matter. Use XMLEXISTS when possible to help the optimizer. If no suitable relational index is available, then using an XML index requires the XMLEXISTS function in the vast majority of cases.

Although using predicates to filter rows is a good practice in any situation, because XMLQUERY and XMLTABLE can be more CPU-heavy than other built-in functions, good filtering is critical.

## 11.7.3 Simple XPath expressions

As a generalization, *simple* XPath expressions perform much better than *complex* XPath expressions, because they are more likely to qualify for XML index access path selection.

For example, XPath with forward slash (/) generally performs better than with double forward slash (//), both for queries and XML index specifications.

Always review the latest APARs to make sure that you download PTFs that improve performance of XML processing. The best place to start is II14426, which is the information APAR to link together all the XML support delivery APARs:

http://www.ibm.com/support/docview.wss?uid=isg1II14426

A

# Application scenario documents

Many international organizations publish XML standards for various industries. We are using the Bank To Customer Statement V2, one of the ISO 20022 (Universal financial industry message scheme) as the openly published XML standard for the XML documents. Our scenario is based on receiving and processing one of those messages from a financial institution such as a bank. You can read about the ISO 20022 Universal financial Industry message scheme at the following address:

http://www.iso20022.org

A full description of the message is in the Payments_Maintenance_2009.pdf document, which you may download from the following address:

http://www.iso20022.org/documents/general/Payments_Maintenance_2009.zip

© Copyright IBM Corp. 2011. All rights reserved.

# A.1 Schema

We have used the Bank To Customer Statement V2 schema, which can be downloaded from the following location:

http://www.iso20022.org/documents/messages/camt/schemas/camt.053.001.02.zip

This schema is also available as additional material to download, as described in Appendix B, “Additional material” on page 277.

# A.2 XML message

The XML message is from the following location and was augmented by adding a second Stmt element that is same as the first one but with minor changes to text values:

http://www.iso20022.org/documents/messages/camt/instances/camt.053.001.02.zip

The updated XML message is available as additional material to download, as described in Appendix B, “Additional material” on page 277.

We provide two copies of the message.

- Example A-1 shows the entire message that is inserted into BK_TO_CUST_STMT table intact.
- The other copy (Example A-2 on page 274) is a shorter version that contains only the elements that we actually process.

*Example A-1 XML message received*

```
<?xml version="1.0" encoding="UTF-8" ?>
  <Document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02">
    <BkToCstmrStmt>
      <GrpHdr>
         <MsgId>AAAASESS-FP-STAT001</MsgId>
         <CreDtTm>2010-10-18T17:00:00+01:00</CreDtTm>
         <MsgPgntn>
          <PgNb>1</PgNb>
           <LastPgInd>true</LastPgInd>
         </MsgPgntn>
      </GrpHdr>
      <Stmt>
         <Id>AAAASESS-FP-STAT001</Id>
         <CreDtTm>2010-10-18T17:00:00+01:00</CreDtTm>
           <FrToDt>
            <FrDtTm>2010-10-18T08:00:00+01:00</FrDtTm>
             <ToDtTm>2010-10-18T17:00:00+01:00</ToDtTm>
           </FrToDt>
           <Acct>
            <Id>
              <Othr>
                <Id>50000000054910000003</Id>
               </Othr>
            </Id>
            <Ownr>
              <Nm>FINPETROL</Nm>
            </Ownr>
            <Svcr>
```

```
   <FinInstnId>
      <Nm>AAAA BANKEN</Nm>
      <PstlAdr>
       <Ctry>SE</Ctry>
      </PstlAdr>
   </FinInstnId>
 </Svcr>
</Acct>
<Bal>
  <Tp>
    <CdOrPrtry>
      <Cd>OPBD</Cd>
    </CdOrPrtry>
  </Tp>
  <Amt Ccy="SEK">500000</Amt>
  <CdtDbtInd>CRDT</CdtDbtInd>
  <Dt>
   <Dt>2010-10-15</Dt>
  </Dt>
</Bal>
<Bal>
  <Tp>
    <CdOrPrtry>
      <Cd>CLBD</Cd>
    </CdOrPrtry>
  </Tp>
  <Amt Ccy="SEK">435678.50</Amt>
  <CdtDbtInd>CRDT</CdtDbtInd>
  <Dt>
   <Dt>2010-10-18</Dt>
  </Dt>
</Bal>
<Ntry>
  <Amt Ccy="SEK">105678.50</Amt>
  <CdtDbtInd>CRDT</CdtDbtInd>
  <Sts>BOOK</Sts>
  <BookgDt>
   <DtTm>2010-10-18T13:15:00+01:00</DtTm>
  </BookgDt>
  <ValDt>
   <Dt>2010-10-18</Dt>
  </ValDt>
  <AcctSvcrRef>AAAASESS-FP-CN_98765/01</AcctSvcrRef>
  <BkTxCd>
   <Domn>
    <Cd>PAYM</Cd>
      <Fmly>
       <Cd>0001</Cd>
        <SubFmlyCd>0005</SubFmlyCd>
      </Fmly>
   </Domn>
  </BkTxCd>
  <NtryDtls>
   <TxDtls>
     <Refs>
       <EndToEndId>MUELL/FINP/RA12345</EndToEndId>
      </Refs>
     <RltdPties>
       <Dbtr>
         <Nm>MUELLER</Nm>
        </Dbtr>
     </RltdPties>
   </TxDtls>
  </NtryDtls>
</Ntry>
<Ntry>
  <Amt Ccy="SEK">200000</Amt>
  <CdtDbtInd>DBIT</CdtDbtInd>
  <Sts>BOOK</Sts>
  <BookgDt>
   <DtTm>2010-10-18T10:15:00+01:00</DtTm>
  </BookgDt>
  <ValDt>
   <Dt>2010-10-18</Dt>
  </ValDt>
  <AcctSvcrRef>AAAASESS-FP-ACCR-01</AcctSvcrRef>
  <BkTxCd>
   <Domn>
     <Cd>PAYM</Cd>
      <Fmly>
       <Cd>0001</Cd>
        <SubFmlyCd>0003</SubFmlyCd>
      </Fmly>
   </Domn>
  </BkTxCd>
  <NtryDtls>
   <Btch>
     <MsgId>FINP-0055</MsgId>
      <PmtInfId>FINP-0055/001</PmtInfId>
      <NbOfTxs>20</NbOfTxs>
    </Btch>
  </NtryDtls>
</Ntry>
<Ntry>
  <Amt Ccy="SEK">30000</Amt>
  <CdtDbtInd>CRDT</CdtDbtInd>
  <Sts>BOOK</Sts>
  <BookgDt>
   <DtTm>2010-10-18T15:15:00+01:00</DtTm>
  </BookgDt>
  <ValDt>
   <Dt>2010-10-18</Dt>
  </ValDt>
  <AcctSvcrRef>AAAASESS-FP-CONF-FX</AcctSvcrRef>
  <BkTxCd>
   <Domn>
     <Cd>TREA</Cd>
      <Fmly>
       <Cd>0002</Cd>
        <SubFmlyCd>0000</SubFmlyCd>
      </Fmly>
   </Domn>
  </BkTxCd>
  <NtryDtls>
   <TxDtls>
      <Refs>
       <InstrId>FP-004567-FX</InstrId>
        <EndToEndId>AAAASS1085FINPSS</EndToEndId>
      </Refs>
      <AmtDtls>
```

```
            <CntrValAmt>
              <Amt Ccy="EUR">3255</Amt>
               <CcyXchg>
                <SrcCcy>EUR</SrcCcy>
                 <XchgRate>0.1085</XchgRate>
               </CcyXchg>
            </CntrValAmt>
           </AmtDtls>
        </TxDtls>
       </NtryDtls>
     </Ntry>
</Stmt>
<Stmt>
  <Id>AAAASESS-FP-STAT002</Id>
    <CreDtTm>2010-10-17T17:00:00+01:00</CreDtTm>
     <FrToDt>
       <FrDtTm>2010-10-17T08:00:00+01:00</FrDtTm>
       <ToDtTm>2010-10-17T17:00:00+01:00</ToDtTm>
     </FrToDt>
    <Acct>
      <Id>
        <Othr>
          <Id>50000000054910000004</Id>
         </Othr>
      </Id>
      <Ownr>
        <Nm>FINPETROL</Nm>
      </Ownr>
      <Svcr>
        <FinInstnId>
          <Nm>AAAB BANKEN</Nm>
           <PstlAdr>
            <Ctry>SE</Ctry>
           </PstlAdr>
        </FinInstnId>
      </Svcr>
    </Acct>
    <Bal>
      <Tp>
        <CdOrPrtry>
          <Cd>OPAV</Cd>
        </CdOrPrtry>
      </Tp>
      <Amt Ccy="SEK">500300</Amt>
       <CdtDbtInd>CRDT</CdtDbtInd>
       <Dt>
        <Dt>2010-10-14</Dt>
       </Dt>
    </Bal>
    <Bal>
      <Tp>
        <CdOrPrtry>
          <Cd>FWAV</Cd>
        </CdOrPrtry>
      </Tp>
      <Amt Ccy="SEK">435478.50</Amt>
       <CdtDbtInd>CRDT</CdtDbtInd>
       <Dt>
        <Dt>2010-10-17</Dt>
       </Dt>
```

```
</Bal>
<Ntry>
  <Amt Ccy="SEK">105378.50</Amt>
   <CdtDbtInd>CRDT</CdtDbtInd>
   <Sts>BOOK</Sts>
   <BookgDt>
    <DtTm>2010-10-17T13:15:00+01:00</DtTm>
   </BookgDt>
  <ValDt>
    <Dt>2010-10-17</Dt>
   </ValDt>
   <AcctSvcrRef>AAAASESS-FP-CN_98764/01</AcctSvcrRef>
   <BkTxCd>
    <Domn>
      <Cd>PAYM</Cd>
       <Fmly>
        <Cd>0002</Cd>
         <SubFmlyCd>0004</SubFmlyCd>
       </Fmly>
    </Domn>
   </BkTxCd>
   <NtryDtls>
    <TxDtls>
      <Refs>
        <EndToEndId>MUELL/FINP/RA12344</EndToEndId>
       </Refs>
      <RltdPties>
        <Dbtr>
          <Nm>MUELLAR</Nm>
         </Dbtr>
      </RltdPties>
    </TxDtls>
   </NtryDtls>
</Ntry>
<Ntry>
  <Amt Ccy="SEK">200100</Amt>
   <CdtDbtInd>DBIT</CdtDbtInd>
   <Sts>BOOK</Sts>
   <BookgDt>
    <DtTm>2010-10-17T10:15:00+01:00</DtTm>
   </BookgDt>
   <ValDt>
    <Dt>2010-10-17</Dt>
   </ValDt>
   <AcctSvcrRef>AAAASESS-FP-ACCR-02</AcctSvcrRef>
   <BkTxCd>
    <Domn>
      <Cd>TREA</Cd>
       <Fmly>
        <Cd>0002</Cd>
         <SubFmlyCd>0004</SubFmlyCd>
       </Fmly>
    </Domn>
   </BkTxCd>
   <NtryDtls>
     <Btch>
      <MsgId>FINP-0056</MsgId>
       <PmtInfId>FINP-0055/002</PmtInfId>
       <NbOfTxs>21</NbOfTxs>
     </Btch>
```

```
           </NtryDtls>
        </Ntry>
        <Ntry>
           <Amt Ccy="SEK">30020</Amt>
           <CdtDbtInd>CRDT</CdtDbtInd>
           <Sts>BOOK</Sts>
           <BookgDt>
            <DtTm>2010-10-17T15:15:00+01:00</DtTm>
           </BookgDt>
           <ValDt>
            <Dt>2010-10-17</Dt>
           </ValDt>
           <AcctSvcrRef>AAAASESS-FP-CONF-FY</AcctSvcrRef>
           <BkTxCd>
             <Domn>
              <Cd>TREA</Cd>
               <Fmly>
                <Cd>0003</Cd>
                 <SubFmlyCd>0001</SubFmlyCd>
               </Fmly>
             </Domn>
           </BkTxCd>
           <NtryDtls>
            <TxDtls>
              <Refs>
                 <InstrId>FP-004568-FX</InstrId>
                 <EndToEndId>AAAASS1084FINPSS</EndToEndId>
              </Refs>
              <AmtDtls>
                <CntrValAmt>
                  <Amt Ccy="EUR">3254</Amt>
                   <CcyXchg>
                    <SrcCcy>EUR</SrcCcy>
                     <XchgRate>0.1084</XchgRate>
                   </CcyXchg>
                </CntrValAmt>
              </AmtDtls>
            </TxDtls>
           </NtryDtls>
        </Ntry>
    </Stmt>
  </BkToCstmrStmt>
</Document>
```

Example A-2 is a shorter version of the message received and contains only the elements that we actually process in our scenario.

*Example A-2 XML message parts processed*

```
<?xml version="1.0" encoding="UTF-8" ?>
  <Document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:iso:std:iso:20022:tech:xsd:camt.053.001.02">
    <BkToCstmrStmt>
      <GrpHdr>
         <MsgId>AAAASESS-FP-STAT001</MsgId>
         <CreDtTm>2010-10-18T17:00:00+01:00</CreDtTm>
      </GrpHdr>
      <Stmt>
         <Id>AAAASESS-FP-STAT001</Id>
         <CreDtTm>2010-10-18T17:00:00+01:00</CreDtTm>
           <FrToDt>
             <FrDtTm>2010-10-18T08:00:00+01:00</FrDtTm>
             <ToDtTm>2010-10-18T17:00:00+01:00</ToDtTm>
           </FrToDt>
           <Acct>
             <Id>
               <Othr>
                <Id>50000000054910000003</Id>
               </Othr>
            </Id>
            <Ownr>
              <Nm>FINPETROL</Nm>
            </Ownr>
            <Svcr>
              <FinInstnId>
                 <Nm>AAAA BANKEN</Nm>
              </FinInstnId>
            </Svcr>
           </Acct>
           <Bal>
             <Tp>
               <CdOrPrtry>
                 <Cd>OPBD</Cd>
               </CdOrPrtry>
             </Tp>
             <Amt Ccy="SEK">500000</Amt>
             <CdtDbtInd>CRDT</CdtDbtInd>
           </Bal>
           <Bal>
             <Tp>
               <CdOrPrtry>
                 <Cd>CLBD</Cd>
               </CdOrPrtry>
             </Tp>
             <Amt Ccy="SEK">435678.50</Amt>
             <CdtDbtInd>CRDT</CdtDbtInd>
           </Bal>
           <Ntry>
             <Amt Ccy="SEK">105678.50</Amt>
             <BookgDt>
              <DtTm>2010-10-18T13:15:00+01:00</DtTm>
             </BookgDt>
             <AcctSvcrRef>AAAASESS-FP-CN_98765/01</AcctSvcrRef>
           </Ntry>
           <Ntry>
```

```
       <Amt Ccy="SEK">200000</Amt>
       <BookgDt>
        <DtTm>2010-10-18T10:15:00+01:00</DtTm>
       </BookgDt>
       <AcctSvcrRef>AAAASESS-FP-ACCR-01</AcctSvcrRef>
     </Ntry>
     <Ntry>
       <Amt Ccy="SEK">30000</Amt>
       <BookgDt>
        <DtTm>2010-10-18T15:15:00+01:00</DtTm>
       </BookgDt>
       <AcctSvcrRef>AAAASESS-FP-CONF-FX</AcctSvcrRef>
     </Ntry>
</Stmt>
<Stmt>
    <Id>AAAASESS-FP-STAT002</Id>
    <CreDtTm>2010-10-17T17:00:00+01:00</CreDtTm>
     <FrToDt>
       <FrDtTm>2010-10-17T08:00:00+01:00</FrDtTm>
       <ToDtTm>2010-10-17T17:00:00+01:00</ToDtTm>
     </FrToDt>
    <Acct>
      <Id>
        <Othr>
          <Id>50000000054910000004</Id>
         </Othr>
      </Id>
      <Ownr>
        <Nm>FINPETROL</Nm>
      </Ownr>
      <Svcr>
        <FinInstnId>
          <Nm>AAAB BANKEN</Nm>
        </FinInstnId>
      </Svcr>
    </Acct>
    <Bal>
      <Tp>
        <CdOrPrtry>
          <Cd>OPAV</Cd>
        </CdOrPrtry>
      </Tp>
      <Amt Ccy="SEK">500300</Amt>
       <CdtDbtInd>CRDT</CdtDbtInd>
    </Bal>
    <Bal>
      <Tp>
        <CdOrPrtry>
          <Cd>FWAV</Cd>
        </CdOrPrtry>
      </Tp>
      <Amt Ccy="SEK">435478.50</Amt>
       <CdtDbtInd>CRDT</CdtDbtInd>
    </Bal>
    <Ntry>
       <Amt Ccy="SEK">105378.50</Amt>
       <CdtDbtInd>CRDT</CdtDbtInd>
       <ValDt>
        <Dt>2010-10-17</Dt>
       </ValDt>
```

```
           <AcctSvcrRef>AAAASESS-FP-CN_98764/01</AcctSvcrRef>
        </Ntry>
        <Ntry>
           <Amt Ccy="SEK">200100</Amt>
           <ValDt>
            <Dt>2010-10-17</Dt>
           </ValDt>
           <AcctSvcrRef>AAAASESS-FP-ACCR-02</AcctSvcrRef>
        </Ntry>
        <Ntry>
           <Amt Ccy="SEK">30020</Amt>
           <BookgDt>
            <DtTm>2010-10-17T15:15:00+01:00</DtTm>
           </BookgDt>
           <AcctSvcrRef>AAAASESS-FP-CONF-FY</AcctSvcrRef>
        </Ntry>
    </Stmt>
  </BkToCstmrStmt>
</Document>
```

B

# Additional material

This book refers to additional material that can be downloaded from the Internet as described in the following sections.

## Locating the web material

The web material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. Point your web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG247915`

Alternatively, you can go to the IBM Redbooks website at:

`ibm.com/redbooks`

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks publication number SG24-7915-00.

## Using the web material

The additional web material that accompanies this book includes the following files:

| *File name* | *Description* |
|---|---|
| **Storedproceduresamples.zip** | Code samples for the setup of stored procedures that are used for the implementation of the scenarios |
| **Javasamples.zip** | Code samples and data that are used for the implementation of the scenario in Java |
| **Cobolsamples.zip** | Code samples and data that are used for the implementation of the scenario in COBOL |

© Copyright IBM Corp. 2011. All rights reserved.

## System requirements for downloading the web material

The web material requires the following system configuration:

| | |
|---|---|
| **Hard disk space**: | 100 KB |
| **Operating System**: | Windows |
| **Processor**: | All Intel® and AMD processors that are capable of running the supported Windows operating systems (32-bit and x64-based systems). |
| **Memory**: | 2 GB |

## Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the web material `.zip` file into this folder.

### Storedproceduresamples.zip

The compressed file contains code and data to reproduce the application infrastructure.

#### *Prerequisites*

The prerequisites are as follows:

- DB2 Client for Windows V9.7 Fix Pack 3a
- IBM Data Server Driver for JDBC and SQLJ version 4.9 or later (DB2 Client, V9.7 Fix Pack 3a and later provides this support.)
- Environment setup on server for XML schema registration
- The table BK_TO_CSTMR_STMT with XML column, and other related tables shown in Example 6-1 on page 90.

#### *Sample code*

The sample code files are as follows:

- Stored procedure in Example 6-3 on page 92:

  `STOREXML1.db2`
- Java program to drive STOREXML1():

  `Teststorexml1.java`
- Script to run Teststorexml1 java program:

  `callstorexml1.bat`
- Stored procedure in Example 6-4 on page 94:

  `STOREXML2.db2`
- Java program to drive STOREXML2():

  `Teststorexml2.java`
- Script to run Teststorexml2 Java program:

  `callstorexml2.bat`
- Stored procedure to prime an MQ queue with an XML message:

  `LOADMQ.db2`

- Stored procedure in Example 6-7 on page 98:

  `STOREXML3.db2`
- Stored procedure in Example 6-11 on page 100:

  `STOREXML4.db2`
- Stored procedure in Example 6-25 on page 117:

  `STOREXML5.db2`
- Java program to drive STOREXML5():

  `Teststorexml5.java`
- Script to run Teststorexml5 java program:

  `callstorexml5.bat`
- Stored procedure in Example 6-34 on page 126:

  `RECEIVE_CDC.db2`
- Java program to drive RECEIVE_CDC():

  `Testcdc.java`
- Script to run Testcdc Java program:

  `callcdc.bat`

### *Sample data*

The sample data files are as follows:

- DDL for all samples:

  `XMLTABLES.DDL`
- Test data for all samples:

  `XML_TEST_SOURCE.SQL`
- Script to reset test data for all samples:

  `RESET_TEST_SOURCE.SQL`

## Javasamples.zip

The compressed file contains code and data to reproduce our Java scenario.

### *Prerequisites*

The prerequisites are as follows:

- SDK for Java Version 6 or later
- IBM Data Server Driver for JDBC and SQLJ version 4.9 or later (DB2 Client, V9.7 Fix Pack 3a and later provides this support.)
- Environment setup on server for XML schema registration
- The table BK_TO_CSTMR_STMT with XML column must be created (see Example 7-5 on page 141).

### *Sample code*

The sample code files are as follows:

- Class that registers the XML schema to the DB2 databases:

  `RegisterSchema.java`

- Class that validates and inserts XML data into db2 table:

  `InsertXML.java`

- Class that demonstrates partial updates of XML documents:

  `UpdateXML.java`

- Class that demonstrates the retrieving entire or partial XML document to a SQLXML object:

  `SelectXML.java`

- Class that creates SQL stored procedure with XML as parameter to shred XML document, then calls the SQL stored procedure from Java:

  `XMLProcedure.java`

- Class that transforms retrieved XML document into an new XML or HTML document:

  `TransformXML.java`

- Class that sends a XML message generated by XSLT to WebSphere MQ:

  `SendMQMessage.java`

### *Sample data*

The sample data files are as follows:

- Schema of bank to customer statement message:

  `camt.053.001.02.xsd`

- XML file of bank to customer statement message:

  ```
  camt.053.001.02.xml
  camt.053.001.03.xml
  camt.053.001.04.xml
  ```

- XSLT file to transform XML message:

  ```
  camt.053.001.04.xsl
  camt.053.001.05.xsl
  ```

- XML document sent to WebSphere MQ:

  `xmloutput.xml`

## Cobolsamples.zip

The compressed file contains code and data to reproduce our COBOL scenario.

### *Prerequisites*

The prerequisites are as follows:

- Enterprise COBOL for z/OS Version 4.1 or later
- DB2 Connect Version 9.5 or later
- Configuration of DB2 for z/OS node in DB2 Connect
- Setup of DB2 XML schema repository

- The XML schema must be registered (see Example 8-8 on page 166).
- The table BK_TO_CSTMR_STMT with XML column must be created (see Example 8-9 on page 166).

### *Sample code*

The sample code files are as follows:

- COBOL program for inserting a BankToCustomerStatement into table BK_TO_CUSTMR_STSMT using a file reference variable:

  `INSBKST.txt`

- COBOL program for selecting a BankToCustomerStatement from table BK_TO_CUSTMR_STSMT into a file using a file reference variable:

  `GETSTMT.txt`

- COBOL program for selecting entries from a BankToCustomerStatement from table BK_TO_CUSTMR_STSMT using XMLTABLE:

  `GETNTRY.txt`

- COBOL program for altering the message recipient of a BankToCustomerStatement in table BK_TO_CUSTMR_STSMT using XMLMODIFY. The message recipient element is received as a file reference variable.

  `UPDRCPT.txt`

- COBOL program for altering the message recipient of a BankToCustomerStatement in table BK_TO_CUSTMR_STSMT using XMLMODIFY. The message recipient element is generated using COBOL XML GENERATE statement.

  `UPDRCP2.txt`

- JCL for precompiling, compiling, link-editing and binding COBOL program INSBKST. It can be modified to prepare other COBOL programs by replacing INSBKST with other COBOL program name.

  `PREPPROG.txt`

- JCL for running COBOL program INSBKST. It can be modified to run other COBOL programs by replacing the program name, but take care to enter the correct input variables.

  `RUNPROG.txt`

### *Sample data*

The sample data files are as follows:

- Schema for BankToCustomerStatement message:

  `camt.053.001.02.xsd`

- XML file with BankToCustomerStatement message:

  `camt.stmt.xml`

- XML file with MsgRcpt element:

  `camt.msgrcpt.xml`

# Glossary

**A**

**address space.** A range of virtual storage pages identified by a number (ASID) and a collection of segment and page tables which map the virtual pages to real pages of the computer's memory.

**address space connection.** The result of connecting an allied address space to DB2. Each address space containing a task that is connected to DB2 has exactly one address space connection, although more than one task control block (TCB) can be present. See *allied address space* and *task control block*.

**Advanced Program-to-Program Communication (APPC).** (1) The general facility characterizing the LU6.2 architecture and its implementation in separate SNA products. (2) Sometimes used to refer to an LU6.2 product feature, in particular, such as an APPC application programming interface.

**allied address space.** An area of storage external to DB2 that is connected to DB2 and is therefore capable of requesting DB2 services.

**American National Standards Institute (ANSI).** An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

**ANSI.** See *American National Standards Institute*.

**APAR.** See *authorized program analysis report*.

**API.** See *application program interface.*

**applet.** See *Java Applet*.

**application.** (1) A program or set of programs that performs a task, for example, a payroll application. (2) In Java programming, a self-contained, stand-alone Java program that includes a static main method. It does not require an applet viewer. Contrast with applet.

**application plan.** The control structure that is produced during the bind process and used by DB2 to process SQL statements encountered during statement execution.

**application program interface (API).** A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or licensed program.

**application requester (AR).** See *requester*.

**Application Service Provider (ASP).** An ASP is an agent or broker that aggregates, facilitates and brokers IT services to deliver IT-enabled business solutions across a network through subscription-based pricing.

**application-owning region (AOR).** A CICS® region in an MRO environment that “owns” the CICS applications, and invokes them on behalf of remotely attached terminal (or web) users.

**AR.** application requester. See *requester*.

**ASCII.** (1) American Standard Code for Information Interchange. A standard assignment of 7-bit numeric codes to characters. See also *Unicode*. (2) An encoding scheme used to represent strings in many environments, typically on PCs and workstations. Contrast with *EBCDIC*.

**attribute.** In XML, a `name="value"` pair that can be placed in the start tag of an element. The value must be quoted with single or double quotation marks.

**authorization ID.** A string that can be verified for connection to DB2 and to which a set of privileges are allowed. It can represent an individual, an organizational group, or a function, but DB2 does not determine this representation.

**authorized program analysis report (APAR).** A report of a problem caused by a suspected defect in a current, unaltered release of a program.

**automatic bind.** (More correctly automatic rebind). A process by which SQL statements are bound automatically (without a user issuing a BIND command) when an application process begins execution and the bound application plan or package it requires is not valid.

**auxiliary table.** A table containing LOB or XML columns and referenced from the base table.

**B**

**base table.** (1) A table created by the SQL CREATE TABLE statement that is used to hold persistent data. Contrast with result table and temporary table. (2) A table containing a LOB or XML column definition. The actual LOB or XML column data is not stored along with the base table. The base table contains a row identifier for each row and an indicator column for each of its LOB or SML columns. Contrast with *auxiliary table*.

© Copyright IBM Corp. 2011. All rights reserved.

**basic mode.** A S/390® central processing mode that does not use logical partitioning. Contrast with *logically partitioned (LPAR) mode*.

**bean.** A definition or instance of a JavaBeans component. See *JavaBeans*.

**binary XML format**. A system of storing XML data in binary, as opposed to text, that facilitates more efficient storage and exchange.

**bind.** The process by which the output from the DB2 precompiler is converted to a usable control structure called a package or an application plan. During the process, access paths to the data are selected and some authorization checking is performed.

**browser.** (1) In VisualAge® for Java, a window that provides information about program elements. There are browsers for projects, packages, classes, methods, and interfaces. (2) An Internet-based tool that lets users browse websites.

**built-in function.** A function incorporated in the supplied DB2 code which can be used in SQL statements.

**bytecode.** Machine-independent code generated by the Java compiler and executed by the Java interpreter.

## C

**call level interface (CLI).** A callable API for database access, which is an alternative to using embedded SQL. In contrast to embedded SQL, DB2 CLI does not require the user to precompile or bind applications, but rather provides a standard set of functions to process SQL statements and related services at run time.

**Cascading Style Sheet (CSS).** CSS defines a stylesheet language for HTML 4.0. A web page designer uses CSS to separately specify style elements of a web page, such as colors, fonts and font styles.

**case-sensitive.** Indicates whether an application, processor, or operating system distinguishes between upper and lower case text. If it does distinguish, it is case-sensitive. XML tags are case-sensitive, but HTML tags are not.

**casting.** Explicitly converting an object or primitive's data type.

**catalog.** In DB2, a collection of tables that contains descriptions of objects such as tables, views, and indexes.

**catalog table.** Any table in the DB2 catalog.

**CGI.** The Common Gateway Interface (CGI) is a means of allowing a web server to execute a program that you provide rather than to retrieve a file. A number of popular web servers support the CGI. For certain applications (for example, displaying information from a database), you must do more than simply retrieve an HTML document from a disk and send it to the web browser. For such applications, the web server has to call a program to generate the HTML to be displayed. The CGI is not the only such interface, however.

**channel-attached.** (1) Pertaining to attachment of devices directly by data channels (I/O channels) to a computer. (2) Pertaining to devices attached to a controlling unit by cables rather than by telecommunication lines.

**character large object (CLOB).** A sequence of bytes representing single-byte characters or a mixture of single and double-byte characters where the size can be up to 2 GB - 1. Although the size of character large object values can be anywhere up to 2 GB - 1, in general, they are used whenever a character string might exceed the limits of the VARCHAR type.

**class.** An encapsulated collection of data and methods to operate on the data. A class may be instantiated to produce an object that is an instance of the class.

**class hierarchy.** The relationships between classes that share a single inheritance. All Java classes inherit from the Object class.

**class method.** Methods that apply to the class as a whole rather than its instances (also called a *static method*).

**class variable.** Variables that apply to the class as a whole rather than its instances (also called a *static field*).

**CLASSPATH.** In your deployment environment, the environment variable keyword that specifies the directories in which to look for class and resource files.

**CLI.** See *call level interface*.

**client.** (1) A networked computer in which the IDE is connected to a repository on a team server. (2) See requester.

**CLOB.** See *character large object*.

**codebase.** An attribute of the <APPLET> tag that provides the relative path name for the classes. Use this attribute when your class files reside in a different directory than your HTML files.

**column function.** An SQL operation that derives its result from a collection of values across one or more rows. Contrast with scalar function.

**commit.** The operation that ends a unit of work by releasing locks so that the database changes made by that unit of work can be perceived by other processes.

**Common Connector Framework.** In the Enterprise Access Builder, interface and class definitions that provide a consistent means of interacting with enterprise resources (for example, CICS and Encina transactions) from any Java execution environment.

**connection.** In the VisualAge for Java Visual Composition Editor, a visual link between two components that represents the relationship between the components. Each connection has a source, a target, and other properties.

**connection handle.** The data object containing information that is associated with a connection that is managed by DB2 CLI. This object includes general status information, transaction status, and diagnostic information.

**content model.** In XML, the expression specifying what elements and data are allowed within an element.

**cookie.** (1) A small file stored on an individual's computer; this file allows a site to tag the browser with a unique identification. When a person visits a site, the site's server requests a unique ID from the person's browser. If this browser does not have an ID, the server delivers one. On the Windows/Intel platform, the cookie is delivered to a `cookies.txt` file; on a Macintosh platform, it is delivered to `MagicCookie`. Just as someone can track the origin of a phone call with Caller ID, companies can use cookies to track information about behavior. (2) Persistent data stored by the client in the Servlet Builder.

**cursor.** A named control structure used by an application program to point to a row of interest within some set of rows, and to retrieve rows from the set, possibly making updates or deletions.

**customer relationship management (CRM).** CRM includes the systems and infrastructure required to analyze, capture and share all parts of the customer's relationship with the enterprise. From a strategy perspective, it represents a process to measure and allocate organizational resources to those activities that have the greatest return and impact on profitable customer relationships.

**D**

**data access bean.** In the VisualAge for Java Visual Composition Editor, a bean that accesses and manipulates the content of JDBC/ODBC-compliant relational databases.

**Data Access Builder.** A VisualAge for Java Enterprise tool that generates beans to access and manipulate the content of JDBC/ODBC-compliant relational databases.

**data source.** A local or remote relational or non-relational data manager that is capable of supporting data access through an ODBC driver which supports the ODBC APIs. In the case of DB2 for OS/390®, the data sources are always relational database managers.

**database management system (DBMS).** A software system that controls the creation, organization, and modification of a database and access to the data stored within it.

**DB2 thread.** The DB2 structure that describes an application's connection, traces its progress, processes resource functions, and delimits its accessibility to DB2 resources and services.

**DBCLOB.** See *double-byte character large object*.

**DBMS.** See *database management system*.

**direct access storage device (DASD).** A mass storage medium on which a computer stores data.

**distributed relational database architecture (DRDA).** A connection protocol for distributed relational database processing that is used by IBM relational database products. DRDA includes protocols for communication between an application and a remote relational database management system, and for communication between relational database management systems.

**DLL.** See *dynamic link library.*

**Document Object Model (DOM).** DOM allows the representation and manipulation of an XML document in memory as a programming object. DOM is defined by the World Wide Web Consortium.

**document type definition (DTD).** A DTD is a definition of which Elements and Attributes are acceptable in a specific XML file. The DTD therefore defines a subset of XML which may be used for a particular application.

**DOM.** See *Document Object Model*.

**DOM Tree.** A DOM Tree is an in-memory representation of an XML Document.

**double precision.** A floating-point number that contains 64 bits. See also *single precision*.

**double-byte character large object (DBCLOB).** A sequence of bytes representing double-byte characters where the size can be up to 2 GB. Although the size of double-byte character large object values can be anywhere up to 2 GB, in general, they are used when a double-byte character string might exceed the limits of the VARGRAPHIC type.

**DRDA.** See distributed relational database architecture.

**duplex.** Pertaining to communication in which data or control information can be sent and received at the same time. Contrast with *half duplex.*

**dynamic bind.** A process by which SQL statements are bound as they are entered.

**Dynamic I/O Reconfiguration.** A S/390 function that allows I/O configuration changes to be made nondisruptively to the current operating I/O configuration.

**dynamic link library (DLL).** A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a DLL can be shared simultaneously by several applications. The DLL's Enterprise Access Builders also generate platform-specific DLLs for the workstation and OS/390 platforms.

**dynamic SQL.** SQL statements that are prepared and executed within an application program while the program is executing. In dynamic SQL, the SQL source is contained in host language variables rather than being coded into the application program. The SQL statement can change several times during the application program's execution.

**E**

**EBCDIC.** Extended Binary Coded Decimal Interchange Code. An encoding scheme used to represent character data in the MVS™, VM, VSE, and OS/400® environments. Contrast with ASCII.

**EBNF.** Extended Backus-Naur Form. A formal set of production rules that comprise a grammar defining another language, such as XML.

**electronic data interchange (EDI).** The automatic machine-to-machine transfer of trading documents (for example, invoices and purchase orders) using electronic networks such as the Internet. Originally conducted only through value-added networks, EDI is gradually moving to the Internet.

**element.** In XML, a start tag and its end tag, plus the content between the tags. An empty tag is also an element.

**embedded SQL.** SQL statements coded within an application program. See *static SQL*.

**embedded Java.** An API and application environment for high-volume embedded devices, such as mobile phones, pagers, process control, instrumentation, office peripherals, network routers and network switches. The embedded Java applications run on real-time operating systems and are optimized for the constraints of small-memory footprints and diverse visual displays.

**empty declaration.** In XML, the DTD declaration for an empty tag. For example, if `<abcd/>` is an empty tag, the empty declaration looks like: `<!ELEMENT abcd EMPTY>`.

**empty tag.** In XML, a start and end tag combined in one tag. The tag has a trailing slash, so an XML parser can immediately recognize it as an empty tag and not bother looking for a matching end tag. For example, if `abcd` is an empty tag, it looks like `<abcd/>`.

**Enterprise Java.** Includes Enterprise JavaBeans and also open API specifications for: database connectivity, naming and directory services, CORBA/IIOP interoperability, pure Java distributed computing, messaging services, managing system and network resources, and transaction services.

**Enterprise JavaBeans (EJB).** A cross-platform component architecture for the development and deployment of multitier, distributed, scalable, object-oriented Java applications. The Enterprise JavaBeans specification defines a way of building transactionally aware business objects in Java.

**Enterprise Systems Architecture/390 (ESA/390).** An IBM architecture for mainframe computers and peripherals. Processors that follow this architecture include the S/390 Server family of processors.

**entity.** In XML, an entity declaration provides the ability to have constants or replacement strings, which are expanded by a pre-processor. An entity declaration maps some token to a replacement string. Later the token can be prefixed with the ampersand (`&`) character and the replacement string is put in its place.

**environment handle.** In DB2 ODBC, the data object that contains global information regarding the state of the application. An environment handle must be allocated before a connection handle can be allocated. Only one environment handle can be allocated per application.

**ESA/390.** See *Enterprise Systems Architecture/390*.

**exception.** An exception is an object that has caused some sort of new condition, such as an error. In Java, *throwing* an exception means passing that object to an interested party; a signal indicates what kind of condition has taken place. *Catching* an exception means receiving the sent object. *Handling* this exception usually means taking care of the problem after receiving the object, although it might mean doing nothing (which would be bad programming practice).

**executable content.** Code that runs from within an HTML file (such as an applet).

**extends.** A subclass or interface extends a class or interface if it adds fields or methods, or overrides its methods.

**external function.** A function for which the body is written in a programming language that takes scalar argument values and produces a scalar result for each invocation. Contrast with sourced function and built-in function.

**extranet.** In some cases, intranets have connections to other independent intranets. An example is one company connecting its intranet to the intranet of one of its suppliers. Such a connection of intranets is called an extranet. Depending on the implementation, an extranet might be fully or partially visible to the outside.

**F**

**factory.** A bean that dynamically creates instances of beans.

**FastCGI.** FastCGI is a way of combining the advantages of CGI programming with some of the performance benefits you get by using the GWAPI. FastCGI, written by Open Market, Inc., is an extension to normal web server processing. It requires server-specific API support, which is available for AIX®, Sun Solaris, HP-UX, and OS/390. With FastCGI you can start applications in independent address spaces and pass requests for these applications from the web server. The communication is through either the TCP/IP sockets interface or UNIX Domain socket bind path in the hierarchical file system (HFS).

**fibre channel standard.** An ANSI standard for a computer peripheral interface. The I/O interface defines a protocol for communication over a serial interface that configures attached units to a communication fabric. The protocol has four layers. The lower of the four layers defines the physical media and interface, the upper of the four layers defines one or more logical protocols (for example, FCP for SCSI command protocols and FC-SB-2 for FICON® for ESA/390). Refer to ANSI X3.230.1999x.

**FICON.** (1) An ESA/390 computer peripheral interface. The I/O interface uses ESA/390 logical protocols over a FICON serial interface that configures attached units to a FICON communication fabric. (2) An FC4 proposed standard that defines an effective mechanism for the export of the SBCON command protocol through fibre channels.

**field.** A data object in a class; for example, a variable.

**File Transfer Protocol (FTP).** In the Internet suite of protocols, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

**first tier.** The client, the hardware and software with which the user interacts.

**foreign key.** A key that is specified in the definition of a referential constraint. Because of the foreign key, the table is a dependent table. The key must have the same number of columns, with the same descriptions, as the primary key of the parent table.

**form data.** A generated class that represents the HTML form elements in a visual servlet.

**FTP.** See *File Transfer Protocol*.

**function.** A specific purpose of an entity or its characteristic action, such as a column function or scalar function. See *column function* and *scalar function*. Furthermore, functions can be user-defined, built-in, or generated by DB2. See *user-defined function*, *external function*, *sourced function*.

**G**

**garbage collection.** Java's ability to automatically clean up inaccessible unused memory areas. Garbage collection slows performance, but keeps the machine from running out of memory.

**GWAPI.** Because CGI has several architectural limitations, most web servers provide an equivalent mechanism that is optimized for their native environment. Domino® Go Web Server, IBM's strategic web server, offers the Domino Go Web Server Application Programming Interface (GWAPI), optimized for a given environment, such as OS/390. The GWAPI enables you to create dynamic content similar to the CGI, but in a more specialized way than the generalized CGI. The GWAPI process is similar to OS/390 exit processing. There is an exit point for various server functions that can be exploited.

**H**

**half duplex.** In data communication, pertaining to transmission in only one direction at a time. Contrast with *duplex.*

**handle.** In DB2 CLI, a variable that refers to a data structure and associated resources. See *connection handle*, *environment handle*.

**hard disk drive.** (1) A storage media within a storage server used to maintain information that the storage server requires. (2) A mass storage medium for computers that is typically available as a fixed disk or a removable cartridge.

**hierarchy.** The order of inheritance in object-oriented languages. Each class in the hierarchy inherits attributes and behavior from its superclass, except for the top-level Object class.

**HTTPS.** HTTPS is a de facto standard developed by Netscape for making HTTP flows secure. Technically, it is the use of HTTP over SSL.

**Hypertext Markup Language (HTML).** A file format, based on SGML, for hypertext documents on the Internet. Allows for the embedding of images, sounds, video streams, form fields and simple text formatting. References to other objects are embedded using URLs, enabling readers to jump directly to the referenced document.

**Hypertext Transfer Protocol (HTTP).** The Internet protocol, based on TCP/IP, used to fetch hypertext objects from remote hosts.

**I**

**IDE.** See *Integrated Development Environment*.

**identifier.** A unique name or address that identifies things such as programs, devices or systems.

**initial program load (IPL).** (1) The initialization procedure that causes an operating system to commence operation. (2) The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction. (3) The process of loading system programs and preparing a system to run jobs.

**Integrated Development Environment (IDE).** In VisualAge for Java, the set of windows that provide the user with access to development tools. The primary windows are the Workbench, Log, Console, Debugger, and Repository Explorer.

**Internet.** The vast collection of interconnected networks that use TCP/IP and evolved from the ARPANET of the late 1960s and early 1970s. The number of independent networks connected into this vast global net is growing daily.

**Internet Protocol (IP).** In the Internet suite of protocols, a connectionless protocol that routes data through a network or interconnected networks. IP acts as an intermediary between the higher protocol layers and the physical network. However, this protocol does not provide error recovery and flow control, and does not guarantee the reliability of the physical network.

**interpreter.** A tool that translates and executes code line-by-line.

**Intranet.** A private network inside a company or organization that uses the same kinds of software that you would find on the Internet, but that are only for internal use. As the Internet has become more popular, many of the tools used on the Internet are being used in private networks; for example, many companies have web servers that are available only to employees.

**IP.** See *Internet Protocol*.

**IPL.** See *initial program load*.

**J**

**JAR file format.** JAR (Java Archive) is a platform-independent file format that aggregates many files into one. Multiple Java applets and their requisite components (.class files, images, sounds and other resource files) can be bundled in a JAR file and subsequently downloaded to a browser in a single HTTP transaction.

**Java.** An object-oriented programming language for portable, interpretive code that supports interaction among remote objects. Java was developed and specified by Sun Microsystems, Incorporated. The Java environment consists of the JavaOS, the Virtual Machines for various platforms, the object-oriented Java programming language, and several class libraries.

**Java applet.** A small Java program designed to run within a web browser. It is downloadable and executable by a browser or network computer.

**JavaBeans.** Java's component architecture, developed by Sun, IBM, and others. The components, called Java beans, can be parts of Java programs, or they can exist as self-contained applications. Java beans can be assembled to create complex applications, and they can run within other component architectures (such as ActiveX and OpenDoc).

**Java Development Kit (JDK).** The set of Java technologies made available to licensed developers by Sun Microsystems. Each release of the JDK contains the following: the Java Compiler, Java Virtual Machine, Java Class Libraries, Java Applet Viewer, Java Debugger, and other tools.

**Java Naming and Directory Interface (JNDI).** A set of APIs that assist with the interfacing to multiple naming and directory services. (Definition copyright 1996-1999 Sun Microsystems, Inc. All Rights Reserved. Used by permission.)

**Java Native Interface (JNI).** A native programming interface that allows Java code running inside a Java Virtual Machine (VM) to interoperate with applications and libraries written in other programming languages, such as C and C++.

**Java Platform.** The Java Virtual Machine and the Java Core classes make up the Java Platform. The Java Platform provides a uniform programming interface to a 100% Pure Java program regardless of the underlying operating system. (Definition copyright 1996-1999 Sun Microsystems, Inc. All Rights Reserved. Used by permission.)

**Java Remote Method Invocation (RMI).** Java Remote Method Invocation is method invocation between peers, or between client and server, when applications at both ends of the invocation are written in Java. Included in JDK 1.1.

**Java Runtime Environment (JRE).** A subset of the Java Development Kit for users and developers who want to redistribute the JRE. The JRE consists of the Java Virtual Machine, the Java Core Classes, and supporting files. (Definition copyright 1996-1999 Sun Microsystems, Inc. All Rights Reserved. Used by permission.)

**Java Server Page (JSP).** Java Server Pages are web pages that include dynamic tags which are executed on the server. JSPs are the presentation layer for web-based applications built in Java.

**Java Virtual Machine (JVM).** A software implementation of a central processing unit (CPU) that runs compiled Java code (applets and applications).

**Java Database Connectivity (JDBC).** In the JDK, the specification that defines an API that enables programs to access databases that comply with this standard.

**JavaDoc.** Sun's tool for generating HTML documentation on classes by extracting comments from the Java source code files.

**JavaScript.** A scripting language used within an HTML page. Superficially similar to Java but JavaScript scripts appear as text within the HTML page. Java applets, however, are programs written in the Java language and are called from within HTML pages or run as standalone applications.

**JDBC.** See *Java Database Connectivity.*

**JIT.** See *Just-In-Time compiler.*

**JNDI.** See *Java Naming and Directory Interface.*

**JNI.** See *Java Native Interface.*

**JRE.** See *Java Runtime Environment*.

**Just-In-Time compiler (JIT).** A platform-specific software compiler often contained within JVMs. JITs compile Java bytecodes automatically into native machine instructions, thereby reducing the need for interpretation.

**JVM.** See *Java Virtual Machine*.

**L**

**LAN.** See *local area network*.

**large object (LOB).** See *LOB*.

**licensed internal code (LIC).** Microcode that IBM does not sell as part of a machine, but instead, licenses to the customer. LIC is implemented in a part of storage that is not addressable by user programs. Some IBM products use it to implement functions as an alternate to hard-wire circuitry.

**linker.** A computer program for creating load modules from one or more object modules or load modules by resolving cross references among the modules and, if necessary, adjusting addresses. In Java, the linker creates an executable from compiled classes.

**load module.** A program unit that is suitable for loading into main storage for execution. The output of a linkage editor.

**LOB (large object).** A sequence of bytes representing bit data, single-byte characters, double-byte characters, or a mixture of single- and double-byte characters. A LOB can be up to 2 GB -1 bytes in length. See also CLOB, DBCLOB.

**local area network (LAN).** A computer network located in a user’s premises within a limited geographic area.

**logical partition (LPAR).** A set of functions that create a programming environment that is defined by the ESA/390 architecture. ESA/390 architecture uses this term when more than one LPAR is established on a processor. An LPAR is conceptually similar to a virtual machine environment except that the LPAR is a function of the processor. Also, LPAR does not depend on an operating system to create the virtual machine environment.

**logical switch number (LSN).** A two-digit number used by the I/O configuration program (IOCP) to identify a specific ESCON® Director.

**logically partitioned (LPAR) mode.** A central processor mode, available on the Configuration frame when using the PR/SM™ facility, that an operator uses to allocate processor hardware resources among logical partitions. Contrast with *basic mode.*

**LPAR.** See *logical partition*.

## M

**megabyte (MB).** (1) For processor storage, real and virtual storage, and channel volume, 220 or 1,048,576 bytes. (2) For disk storage capacity and communications volumes, 1,000,000 bytes.

**method.** A fragment of Java code within a class that can be invoked and passed a set of parameters to perform a specific task.

**middle tier.** The hardware and software that resides between the client and the enterprise server resources and data. The software includes a web server that receives requests from the client and invokes Java servlets to process these requests. The client communicates with the web server through industry standard protocols such as HTTP and IIOP.

**middleware.** A layer of software that sits between a database client and a database server, making it easier for clients to connect to heterogeneous databases.

**multithreading.** Multiple TCBs executing one copy of code concurrently (sharing a processor) or in parallel (on separate central processors).

## N

**National Committee for Information Technology Standards (NCITS).** NCITS develops national standards, and its technical experts participate, on behalf of the United States in the international standards activities of ISO/IEC JTC 1, information technology.

**native class.** Machine-dependent C code that can be invoked from Java. For multi-platform work, the native routines for each platform need to be implemented.

**native SQL procedure.** An SQL procedure that is processed by converting the procedural statements to a native representation that is stored in the database directory, as is done with other SQL statements. When a native SQL procedure is called, the native representation is loaded from the directory, and DB2 executes the procedure.

**NCITS.** See *National Committee for Information Technology Standards.*

**NUL terminator.** In C, the value that indicates the end of a string. For character strings, the NUL terminator is `X'00'`.

**NUL-terminated host variable.** A varying-length host variable in which the end of the data is indicated by the presence of a NUL terminator.

**null.** A special value that indicates the absence of information.

## O

**object.** The principal building block of object-oriented programs. Objects are software programming modules. Each object is a programming unit consisting of related data and methods.

**ODBC.** See *Open Database Connectivity.*

**ODBC driver.** A DLL that implements ODBC function calls and interacts with a data source.

**Open Database Connectivity (ODBC).** A Microsoft® database API for C that allows access to database management systems by using callable SQL. ODBC does not require the use of an SQL preprocessor. In addition, ODBC provides an architecture in which users add modules called database drivers that link the application to the user's choice of database management systems at run time. Therefore applications no longer need to be directly linked to the modules of all the database management systems that are supported.

**open system.** A system whose characteristics comply with standards made available throughout the industry and that therefore can be connected to other systems complying with the same standards.

**original equipment manufacturers information (OEMI).** A reference to an IBM guideline for a computer peripheral interface. More specifically, refer to IBM S/360 and S/370 Channel to Control Unit Original Equipment Manufacturer's Information. The interface uses ESA/390 logical protocols over an I/O interface that configures attached units in a multi-drop bus environment.

## P

**package.** A program element that contains classes and interfaces.

**partition-by-growth table space**. A universal table space whose size can grow to accommodate data growth. DB2 for z/OS manages partition-by-growth table spaces by automatically adding new data sets when the database needs more space to satisfy an insert operation.

**partitioned-by-range table space.** A type of universal table space that is based on user defined partitioning ranges.

**persistence.** In object models, a condition that allows instances of classes to be stored externally, for example in a relational database.

**Persistence Builder.** In VisualAge for Java, a persistence framework for object models, which enables the mapping of objects to information stored in relational databases and also provides linkages to legacy data on other systems.

**plan.** See *application plan*.

**plan name.** The name of an application plan.

**precompilation.** A processing of application programs containing SQL statements that takes place before compilation. SQL statements are replaced with statements that are recognized by the host language compiler.

**prepare.** The first phase of a two-phase commit process in which all participants are requested to prepare for commit.

**prepared SQL statement.** A named object that is the executable form of an SQL statement that has been processed by the PREPARE statement.

**primary key.** A unique, non-null key that is part of the definition of a table. A table cannot be defined as a parent unless it has a unique key or primary key.

**process.** A program executing in its own address space, containing one or more threads.

**program temporary fix (PTF).** A temporary solution or bypass of a problem diagnosed by IBM in a current unaltered release of a program.

**property.** An initial setting or characteristic of a bean, for example, a name, font, text, or positional characteristic.

**PTF.** See *program temporary fix*.

**R**

**RDBMS.** See *relational database management system*.

**reentrant.** Executable code that can reside in storage as one shared copy for all threads. Reentrant code is not self-modifying and provides separate storage areas for each thread. Reentrancy is a compiler and operating system concept, and reentrancy alone is not enough to guarantee logically consistent results when multithreading.

**reference.** An object's address. In Java, objects are passed by reference rather than by value or by pointers.

**relational database management system (RDBMS).** A relational database manager that operates consistently across supported IBM systems.

**remote.** Refers to any object maintained by a remote DB2 subsystem; that is, by a DB2 subsystem other than the local one. A remote view, for instance, is a view maintained by a remote DB2 subsystem. Contrast with local.

**Remote Method Invocation (RMI).** RMI is a specific instance of the more general term RPC. RMI allows objects to be distributed over the network; that is, a Java program running on one computer can call the methods of an object running on another computer. RMI and java.net are the only 100% pure Java APIs for controlling Java objects in remote systems.

**Remote Object Instance Manager.** In Remote Method Invocation, a program that creates and manages instances of server beans through their associated server-side server proxies.

**Remote Procedure Calls (RPC).** RPC is a generic term referring to any of a series of protocols that are used to execute procedure calls or method calls across a network. RPC allows a program running on one computer to call the services of a program running on another computer.

**requester.** Also application requester (AR). The source of a request to a remote RDBMS, the system that requests the data.

**RMI.** See *Remote Method Invocation.*

**rollback.** The process of restoring data changed by SQL statements to the state at its last commit point. All locks are freed. Contrast with commit.

**RPC.** See *Remote Procedure Calls*.

**runtime system.** The software environment where compiled programs run. Each Java runtime system includes an implementation of the Java Virtual Machine.

**S**

**sandbox.** A restricted environment, provided by the web browser, in which Java applets run. The sandbox offers them services and prevents them from doing anything unacceptable, such as doing file I/O or talking to strangers (servers other than the one from which the applet was loaded). The analogy of applets to children led to calling the environment in which they run the *sandbox*.

**scalar function.** An SQL operation that produces a single value from another value and is expressed as a function name followed by a list of arguments enclosed in parentheses. See also *column function*.

**Secure Sockets Layer (SSL).** SSL is a security protocol that allows communications between a browser and a server to be encrypted and secure. SSL prevents eavesdropping, tampering, or message forgery on your Internet or intranet network.

**security.** Features in Java that prevent web-downloaded applets downloaded from deliberately or inadvertently doing damage. One such feature is the digital signature, which ensures that an applet came unmodified from a reputable source.

**serialization.** Converting an object to a stream, and back again.

**server.** The computer that hosts the web page that contains an applet. The `.class` files that make up the applet, and the HTML files that reference the applet reside on the server. When someone on the Internet connects to a web page that contains an applet, the server delivers the `.class` files over the Internet to the client that made the request. The server is also known as the originating host.

**server bean.** The bean that is distributed using RMI services and is deployed on a server.

**servlet.** See *Java servlet*.

**SGML.** See *Standardized Generalized Markup Language.*

**shell.** The user interface of UNIX system software. In z/OS, an xpg4.2-compliant shell is used. Very often, OMVS is used as an interface for z/OS shells.

**single precision.** A floating-point number that contains 32 bits. See also double precision.

**Small Computer System Interface (SCSI).** (1) An ANSI standard for a logical interface to computer peripherals and for a computer peripheral interface. The interface uses a SCSI logical protocol over an I/O interface that configures attached targets and initiators in a multi-drop bus topology. (2) A standard hardware interface that enables a variety of peripheral devices to communicate with one another.

**SmartGuide.** In IBM software products, an active form of help that guides you through common tasks.

**source type.** An existing type that is used to internally represent a distinct type.

**sourced function.** A function that is implemented by another built-in or user-defined function already known to the database manager. This function can be a scalar function or a column (aggregating) function; it returns a single value from a set of values (for example, MAX or AVG). Contrast with *external function*.

**SQL.** See *Structured Query Language*.

**SSL.** See *Secure Sockets Layer*.

**Standardized Generalized Markup Language.** An ISO/ANSI/ECMA standard that specifies a way to annotate text documents with information about types of sections of a document.

**static bind.** A process by which SQL statements are bound after they have been precompiled. All static SQL statements are prepared for execution at the same time. Contrast with dynamic bind.

**static SQL.** SQL statements, embedded within a program, that are prepared during the program preparation process (before the program is executed). After being prepared, the SQL statement does not change (although values of host variables specified by the statement might change).

**stored procedure.** A user-written application program, that can be invoked through the use of the SQL CALL statement.

**Structured Query Language (SQL).** A standardized language for defining and manipulating data in a relational database. A language used by database engines and servers for data acquisition and definition.

**Sysout.** The regular output for a program on z/OS is SYSOUT. It is the functional equivalent of stdout on UNIX. In batch, there can be multiple SYSOUTs.

**System.** A single instance of the z/OS or OS/390 operating system in a sysplex.

**System Management End User Interface (SMEUI).** A Windows-based tool that makes it possible to perform administrative tasks for WebSphere Application Server from a Windows workstation. The SMEUI tool is used to deploy a new application to WebSphere on z/OS.

**T**

**task control block (TCB).** A control block used to communicate information about tasks within an address space that are connected to DB2. An address space can support many task connections (as many as one per task), but only one address space connection. A TCB manages dispatchable tasks. Each UNIX thread is assigned to a TCB. See *address space connection*.

**TCB.** See *task control block*.

**Telnet.** Telnet provides a virtual terminal facility for users of one computer to act as though they were using a terminal connected to another computer. The Telnet client program communicates with the Telnet daemon on the target system to provide the connection and session.

**temporary table.** A table created by the SQL CREATE GLOBAL TEMPORARY TABLE statement that is used to hold temporary data. Contrast with result table.

**textual XML format.** A system of storing XML data in text, as opposed to binary, that allows for direct human reading.

**thin client.** Thin client usually refers to a system that runs on a resource-constrained machine or that runs a small operating system. Thin clients do not require local system administration, and they execute Java applications delivered over the network.

**third tier.** The third tier, or back end, is the hardware and software that provides database and transactional services. These back-end services are accessed through connectors between the middle-tier web server and the third-tier server. Though this conceptual model depicts the second and third tier as two separate machines, the NCF model supports a logical three-tier implementation in which the software on the middle and third tier is on the same box.

**thread.** A separate flow of control within a program.

**timestamp.** A seven-part value that consists of a date and time expressed in years, months, days, hours, minutes, seconds, and microseconds.

**trace.** A facility that provides the ability to monitor and collect monitoring, auditing, performance, accounting, statistics, and serviceability data.

**Trading communities.** Trading communities bring together buyers and sellers in a central online location to trade, using various online mechanisms including auctions and exchanges, in addition to industry content and application services. Trading communities are owned and operated by both large industry players in closed trading networks and by neutral parties in more fragmented open communities.

**transaction.** (1) In a CICS program, an event that queries or modifies a database that resides on a CICS server. (2) In the Persistence Builder, a representation of a path of code execution. (3) The code activity necessary to manipulate a persistent object. For example, a bank application might have a transaction that updates a company account.

**U**

**UDF.** See *user-defined function.*

**UDT.** See *distinct type*.

**Unicode.** A 16-bit international character set defined by ISO 10646. See also *ASCII.*

**Uniform Resource Identifier (URI)**. Uniquely defines a location on the web. Compared to URL, the term URI is a more general term that also incorporates other schemes for identifying resources.

**Uniform Resource Locator (URL).** Uniquely defines a location on the web. The unique address that tells a browser how to find a specific web page or file. URLs are familiar to anyone who browses the web (for example `http://www.ibm.com`).

**universal table space.** A table space that is both segmented and partitioned.

**URI.** See *Uniform Resource Identifier.*

**URL.** See *Uniform Resource Locator.*

**user-defined data type (UDT).** See *distinct type*.

**user-defined function (UDF).** A function defined to DB2 using the CREATE FUNCTION statement that can be referenced thereafter in SQL statements. A user-defined function can be either an external function or a sourced function. Contrast with *built-in function*.

**V**

**valid.** An XML document is valid if its content conforms to the rules in its DTD.

**variable.** (1) An identifier that represents a data item whose value can be changed while the program is running. The values of a variable are restricted to a certain data type. (2)A data element that specifies a value that can be changed. A COBOL elementary data item is an example of a variable. Contrast with constant.

**vi.** A popular UNIX editor. It can only be used from an ASCII Telnet connection.

**virtual machine.** A software or hardware implementation of a central processing unit (CPU) that manages the resources of a machine and can run compiled code. See *Java Virtual Machine*.

**visual bean.** In the Visual Composition Editor, a bean that is visible to the user in the graphical user interface.

**W**

**Wireless Application Protocol (WAP).** Offers Internet browsing from wireless handsets.

**web.** See *World Wide Web.*

**web application.** A web application is a collection of static pages, JSPs, and servlets that share a common URL prefix, and together make a complete application, accessed over the Internet.

**web browser.** The web uses a client/server processing model. The web browser is the client component. Examples of web browsers include Mozilla, Google, and Chrome. The web browser is responsible for formatting and displaying information, interacting with the user, and invoking external functions, such as Telnet, or external viewers for data types that it does not directly support.

**web server.** Web servers are responsible for servicing requests for information from web browsers. The information can be a file retrieved from the server's local disk or generated by a program called by the server to perform a specific application function. web servers are sometimes referred to as HTTPD servers or daemons. A number of web servers are available for most platforms including most UNIX variants, OS/2 Warp, OS/390, and Windows NT®.

**well-formed.** An XML document is well-formed if there is one root element, and all its child elements are properly nested within each other. Start tags must have end tags, and each empty tag must be designated as such with a trailing slash. Also, all attributes must be quoted, and all entities must be declared.

**white-space.** In XML, white space consists of characters that are not visible, but used in formatting documents or programs. These characters include the space, tab, newline, and carriage-return characters.

**World Wide Web.** A network of servers that contain programs and files. Many of the files contain hypertext links to other documents available through the network.

**WWW.** See *World Wide Web*.

**X**

**XML.** The Extensible Markup Language (XML) is an important emerging standard for structured documents on the web. XML extends HTML beyond a limited tag set and adapts SGML, helping developers to more easily write programs that process this markup and providing for a rich, more complex encoding of information.

**XML attribute**. A name-value pair within a tagged XML element that modifies certain features of the element.

**XML column.** A column of a table that stores XML values and is defined using the data type XML. The XML values that are stored in XML columns are internal representations of well-formed XML documents.

**XML data type.** A data type for XML values.

**XML element.** A logical structure in an XML document that is delimited by a start and an end tag. Anything between the start tag and the end tag is the content of the element.

**XML index.** An index on an XML column that provides efficient access to nodes within an XML document by providing index keys that are based on XML patterns.

**XML lock.** A column-level lock for XML data. The operation of XML locks is similar to the operation of LOB locks.

**XML node.** The smallest unit of valid, complete structure in a document. For example, a node can represent an element, an attribute, or a text string.

**XML node ID index.** An implicitly created index, on an XML table that provides efficient access to XML documents and navigation among multiple XML data rows in the same document.

**XML pattern.** A slash-separated list of element names, an optional attribute name (at the end), or kind tests, that describe a path within an XML document in an XML column. The pattern is a restrictive form of path expressions, and it selects nodes that match the specifications. XML patterns are specified to create indexes on XML columns in a database.

**XML publishing function.** A function that returns an XML value from SQL values. An XML publishing function is also known as an XML constructor

**XML schema.** In XML, a mechanism for describing and constraining the content of XML files by indicating which elements are allowed and in which combinations. XML schemas are an alternative to DTDs and can be used to extend functionality in the areas of data typing, inheritance, and presentation.

**XML schema repository (XSR).** A repository that allows the DB2 database system to store XML schemas. When registered with the XSR, these objects have a unique identifier and can be used to validate XML instance documents.

**XML serialization function.** A function that returns a serialized XML string from an XML value.

**XML table.** An auxiliary table that is implicitly created when an XML column is added to a base table. This table stores the XML data, and the column in the base table points to it.

**XML table space.** A table space that is implicitly created when an XML column is added to a base table. The table space stores the XML table. If the base table is partitioned, one partitioned table space exists for each XML column of data.

**XSL.** Extensible Stylesheet Language. Defines stylesheets for XML Documents. It is composed of two parts: the formatting objects, and XSLT. XSL is defined by the World Wide Web Consortium.

**XSLT.** Extensible Stylesheet Language Transformations. The XSLT defines the part of the XSL specification that allows the stylesheet to reformat and reorganize the XML data. It is most often used to transform XML into XSL.

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

## IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 298. Note that some of the documents referenced here might be available in softcopy only.

- *DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond,* SG24-7604
- *DB2 10 for z/OS Technical Overview,* SG24-7892
- *XML on z/OS and OS/390: Introduction to a Service-Oriented Architecture*, SG24-6826
- *XML Processing on z/OS,* SG24-7810

## Other publications

These publications are also relevant as further information sources:

- *DB2 10 for z/OS Application Programming and SQL Guide,* SC19-2969
- *DB2 10 for z/OS Application Programming Guide and Reference for Java,* SC19-2970
- *DB2 10 for z/OS Installation and Migration Guide,* GC219-2974
- *DB2 10 for z/OS pureXML Guide,* SC19-2981
- *DB2 10 for z/OS SQL Reference,* SC19-2983
- *DB2 pureXML Cookbook,* Matthias Nicola and PAV Kumar-Chatterjee, IBM Press, ISBN-13: 978-0-13-815047-1
- *WebSphere MQ Application Programming Guide Version 6.0,* SC34-6595
- *WebSphere MQ Using Java Version 6.0*, SC34-6591

## Online resources

These websites are also relevant as further information sources:

- Tools and XML functionality for DB2 pureXML users

  `https://www.ibm.com/developerworks/data/library/techarticle/dm-1012xmltools/`
- Extensible Dynamic Binary XML, Client/Server Binary XML Format (XDBX) Version 1.0

  `http://www.ibm.com/support/docview.wss?uid=swg27019354&aid=1`
- XSL Transformations (XSLT) Version 1.0

  `http://www.w3.org/TR/xslt`
- pureXML Devotees

  `http://www.ibm.com/developerworks/wikis/display/db2xml/devotee`

© Copyright IBM Corp. 2011. All rights reserved.

- ISO 3166 code lists

  http://www.iso.org/iso/country_codes/iso_3166_code_lists.htm

- OASIS

  http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl

- ISO 20022 Universal financial industry message scheme

  http://www.iso20022.org/

- Catalogue of ISO 20022 messages

  http://www.iso20022.org/catalogue_of_unifi_messages.page

- XML

  http://www.w3.org/XML

# How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, and also order hardcopy Redbooks publications, at this website:

**ibm.com**/redbooks

# Help from IBM

IBM Support and downloads

**ibm.com**/support

IBM Global Services

**ibm.com**/services

# Index

© Copyright IBM Corp. 2011. All rights reserved.

IBM

Redbooks

# Extremely pureXML in DB2 10 for z/OS

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

# Extremely pureXML in DB2 10 for z/OS

**Develop Java and COBOL applications accessing XML and SQL data**

**Administer your XML and SQL data**

**Choose the best options for installation and use**

The DB2 pureXML feature offers sophisticated capabilities to store, process and manage XML data in its native hierarchical format. By integrating XML data intact into a relational database structure, users can take full advantage of DB2's relational data management features.

In this IBM Redbooks publication, we document the steps for the implementation of a simple but meaningful XML application scenario. We have chosen to provide samples in COBOL and Java language. The purpose is to provide an easy path to follow to integrate the XML data type for the traditional DB2 for z/OS user.

We also add considerations for the data administrator and suggest best practices for ease of use and better performance.

**INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION**

**BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**
**ibm.com**/redbooks

SG24-7915-00 ISBN 0738435147