

Oracle Database 12c: Use XML DB

Student Guide - Volume I

D81146GC10
Edition 1.0
October 2013
D84300

ORACLE®

Author

Lauran K. Serhal

**Technical Contributors
and Reviewers**

Amitabh James Hans

Anjulapponni.A.S

Asha Tarachandani

Beda Hammerschmidt

Dan Melinger

Drew Adams

Geeta Arora

Hui Chang

James Spiller

Joe Greenwald

Lakshmi Narapareddi

Mark Drake

Nancy Greenberg

Qin Yu

Ravi Chennoju

Roger Ford

S. Matt Taylor Jr.

Sriram Krishnamurthy

Srividya Tata

Vikas Arora

Viktor Tchemodanov

Ying Lu

Zhen Hua Liu

Editors

Anwesha Ray

Smita Kommini

Malavika Jinka

Graphic Designers

Rajiv Chandrabhanu

Seema Bopaiah

Publishers

Syed Imtiaz Ali

Michael Sebastian

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

Preface

1 Introduction

Objectives	1-2
Lesson Agenda	1-3
Questions About You	1-4
Course Objectives	1-5
Course Prerequisites	1-6
Suggested Course Agenda	1-7
Database Schemas Used in This Course	1-11
The Human Resources (hr) Schema	1-12
The Order Entry (oe) Schema	1-13
The Purchase Order XML Schema, purchaseorder.xsd, Used in This Course	1-14
Appendices Used in This Course	1-15
Class Account Information	1-16
Course Environment	1-17
Entering SQL Statements Using Oracle SQL*Plus in a Terminal Window	1-18
Using XML in Oracle JDeveloper	1-19
Entering SQL Statements and Coding PL/SQL Using Oracle SQL Developer	1-20
Lesson Agenda	1-21
Starting Oracle SQL Developer and Creating a Database Connection	1-22
Creating Database Schema Objects	1-23
Using the SQL Worksheet	1-24
Using the SQL Worksheet Toolbar	1-25
Executing SQL Statements	1-26
Saving SQL Scripts	1-27
Executing Saved Script Files by Using the @ Command	1-28
Executing SQL Scripts	1-29
Creating an Anonymous PL/SQL Block	1-30
Lesson Agenda	1-31
Available Oracle University Courses at: education.oracle.com	1-32
Where to Go for More Information?	1-33
Oracle Database 12c Release 1 (12.1) Documentation	1-34
Additional Resources: Using Oracle By Example (OBE) in the Online Learning Library	1-36

Additional Resources: Oracle Online Library (OLL)	1-37
The Oracle XML DB Home Page: http://www.oracle.com/technetwork/database-features/xmldb/overview/index.html	1-38
Additional XML Useful Web Sites	1-39
Course Scripts in the /home/oracle/labs Folder	1-40
Summary	1-41
Practice 1-1: Overview	1-42
Practice 1-2: Overview	1-43

2 XML Basic Review

Objectives	2-2
What Is XML	2-3
The Difference Between XML and HTML	2-4
XML Documents Form a Tree Structure	2-5
An Example of a Simple XML Document	2-6
XML Elements and Attributes	2-7
XML Markup Syntax Rules for Elements	2-8
XML Syntax Rules	2-9
XML Attributes	2-10
Well-Formed XML Documents	2-11
Document Type Definition (DTD)	2-12
Why Validate an XML Document?	2-13
XML Namespace	2-14
Why Use XML Namespaces?	2-15
Declaring XML Namespaces: The XMLNS Attribute	2-16
XML Namespace Declarations and Default Namespaces: Example	2-17
XML Schema	2-18
XML Schema Document: Example	2-20
Components of an XML Schema	2-21
XML Schema Components: Example	2-22
XML Path Language	2-23
XPath Terminology	2-24
XPath Model As a Tree (Partial Example)	2-25
XPath Model as an XML Document	2-26
XPath Expressions	2-27
Location Path Expression	2-28
Selecting Nodes	2-29
Class Review: Examine the XML Document	2-30
Location Path Expression: Example	2-34
XPath Node Test Types	2-35
Abbreviated XPath Expression Examples: The XML Document	2-36

Abbreviated XPath Expression: Examples 2-37
Abbreviated XPath Expression Examples: The XML Document 2-38
XPath Predicates 2-39
XQuery: Review 2-41
XQuery Terminology 2-42
XQuery Review: books.xml Document Example 2-43
FLWOR Expressions: Review 2-44
Selecting Nodes From an XML Document 2-45
XQuery Basic Syntax Rules 2-46
EXtensible Stylesheet Language (XSL) and XSL Transformations (XSLT) 2-47
Summary 2-48
Practice 2-1: Overview 2-49
Practice 2-2: Overview 2-50

3 Introduction to Oracle XML DB

Objectives 3-2
Oracle XML DB 3-3
Oracle XML DB: Benefits 3-4
Oracle XML DB: Features 3-7
Summary 3-10

4 Storing XML Data in Oracle XML DB

Objectives 4-2
Lesson Agenda 4-3
XMLType: Overview 4-4
Using XMLType 4-5
Declaring an XMLType Table, Column, and Attribute: Examples 4-6
XMLType Storage Models 4-7
XMLType Storage: Binary XML for Schema-Based Storage 4-9
XMLType Storage: Binary XML for Non-Schema-Based Storage 4-10
Binary XML: Advantages 4-11
XMLType Storage: Structured (object-relational) Storage 4-13
XMLType Structured (Object-Relational) Storage: Advantages and Disadvantages 4-15
Specifying Binary XML Storage 4-16
Specifying Binary XML Storage: Examples 4-17
Allowing Non-Schema-Based Storage 4-20
XMLType Storage: Use Case Scenarios 4-21
Choosing an XMLType Storage Model: Data Centric 4-24
Choosing an XMLType Storage Model: Document-Centric 4-26
XMLType Storage Models: Relative Advantages 4-27

Lesson Agenda 4-29
Loading Data into XMLType 4-30
Specifying SQL Constraints with O-R and Binary XML Storages 4-32
Defining Constraints on Binary XML Data 4-33
Quiz 4-35
Summary 4-36
Practice 4: Overview 4-37

5 Using XML Schema with Oracle XML DB

Objectives 5-2
Lesson Agenda 5-3
W3C XML Schema Recommendation: Overview 5-4
XML Schema Support in Oracle Database 12c 5-5
XML Schema and Oracle XML DB 5-6
XMLType and XML Schema 5-8
XML Schema Management 5-10
Registering an XML Schema 5-11
Registering an XML Schema Example: Method 1 5-12
Registering an XML Schema Example: Method 2 5-13
The REGISTERSCHEMA Procedure Parameters 5-14
Storage and Access Infrastructure 5-15
Local and Global XML Schemas 5-16
Deleting an XML Schema 5-17
Parameters for the DELETESCHEMA Procedure 5-18
Registering an XML Schema for Binary XML 5-19
DBMS_XMLSHEMA.purgeSchema() 5-20
Creating XML Schema-Based XMLType Tables 5-21
Creating XML Schema-Based XMLType Tables: Example 5-22
Quiz 5-24
Practice 5-1: Overview 5-25
Practice 5-2: Overview 5-26
Lesson Agenda 5-27
XML Schema Evolution 5-28
Copy-Based XML Schema Evolution 5-29
DBMS_XMLSHEMA.COPYEVOLVE Procedure 5-30
Using DBMS_XMLSHEMA.COPYEVOLVE: Steps 5-31
Copy-Based XML Schema Evolution Example: Steps Overview 5-33
Step 1: Loading the XML Schema into the Oracle XML DB Repository and
 Registering the Schema 5-35
Step 2: Creating the emp_acme_tab Table 5-37
Step 3: Inserting Values Into the emp_acme Table 5-38

Step 4: Querying for the Number of Rows in the emp_acme Table	5-39
Step 5: Adding a New address Element to the Existing XML Schema	
Definition	5-40
Step 6: Evolving the XML Schema	5-41
Step 7: Inserting New Values in the emp_acme Table	5-42
Step 8: Querying the emp_acme Table to See the Changes	5-43
Copy-Based XML Schema Evolution: Guidelines	5-44
Using DBMS_XMLSHEMA.COPYEVOLVE: Disadvantages	5-45
In-Place XML Schema Evolution	5-46
In-Place XML Schema Evolution: Advantages	5-47
Performing In-Place XML Schema Evolution	5-48
Creating an XML Document for the diffXML Parameter	5-50
In-Place XML Schema Evolution: Example	5-51
In-Place XML Schema Evolution Example: Step 1	5-52
In-Place XML Schema Evolution Example: Step 2	5-53
In-Place XML Schema Evolution Example: Step 3	5-54
In-Place XML Schema Evolution Example: Steps 4 and 5	5-55
Using In-Place XML Schema Evolution: Guidelines	5-56
In-Place XML Schema Evolution: Supported Operations	5-57
In-Place XML Schema Evolution: Restrictions	5-59
Quiz	5-61
Summary	5-62
Practice 5-3: Overview	5-63

6 Oracle XML DB Manageability

Objectives	6-2
Agenda	6-3
Oracle XML Schema Annotations	6-4
Common Uses of XML Schema Annotations	6-5
Annotations Methods	6-6
Purchase-Order XML Schema, purchaseOrder.xsd, Before the Annotations	6-7
Annotated Purchase-Order XML Schema, purchaseOrder.xsd	6-8
Annotating an XML Schema by Using DBMS_XMLSHEMA_ANNOTATE	6-10
Annotation Subprogram Parameters	6-11
Some of the Available Oracle XML DB XML Schema Elements Annotations	6-12
Examples of the Most Commonly Used XML Annotations	6-14
DBMS_XMLSHEMA_ANNOTATE Procedure Example:	
SETDEFAULTTABLE	6-15
DBMS_XMLSHEMA_ANNOTATE Procedure Example: setSQLType	6-16
DBMS_XMLSHEMA_ANNOTATE: Example	6-17
Querying a Registered XML Schema to Obtain Annotations	6-19

XML Schema Annotation Guidelines for Object-Relational Storage	6-20
How Oracle XML DB Maps XML Schema-Based XMLType Tables	6-21
Agenda	6-22
DBMS_XMLSTORAGE_MANAGE Package: Overview	6-23
Summary of the DBMS_XMLSTORAGE_MANAGE Package Subprograms	6-24
Quiz	6-26
Summary	6-27
Practice 6-1: Overview	6-28

7 Partitioning XMLType Tables and Columns

Objectives	7-2
Partitioned Tables and Indexes	7-3
Why Partitioning?	7-4
Ordered Collection Tables	7-5
Equipartitioning	7-6
Advantages of Partitioning an OCT	7-7
Partitioning XMLType Data	7-8
Partitioning XMLType Table: During Table Creation	7-9
Maintaining a Partition	7-10
Manual Maintaining of Partitions: Examples	7-11
Partitioning Binary XML Tables	7-12
Using Virtual Columns to Partition Binary XML Tables	7-13
Steps to Partition an XMLTYPE Table Stored as Binary XML	7-14
Partitioning Data Dictionary Views	7-15
XMLIndex Partitioning and Parallelism	7-16
Quiz	7-17
Summary	7-21
Practice 7: Overview	7-22

8 Using XQuery to Retrieve XML Data in Oracle XML DB

Objectives	8-2
Lesson Agenda	8-3
Retrieving XML Content: Overview	8-4
Types of XML Queries	8-5
XQuery: Review	8-7
XQuery Review: books.xml Document Example	8-8
FLWOR Expressions: Review	8-9
Static Type-Checking of XQuery Expressions	8-11
XPath Expressions Verification	8-12
XQuery Support in Oracle Database	8-14
XMLQuery	8-17

URI Scheme oradb: Querying Table or View Data with the XQuery fn:collection Function 8-19
fn:collection Function: Example 8-21
XMLTable 8-22
XMLTable: Example 8-23
XMLTable Versus XMLQuery: Example 8-24
Lesson Agenda 8-25
Querying the Database: Relational Data 8-26
Using XMLQuery to Query Relational Data 8-27
Joining Tables by Using XMLQuery 8-28
Lesson Agenda 8-29
Querying the Database: XMLType Data 8-30
Lesson Agenda 8-33
Querying an XMLType Table With XMLQuery and fn:collection 8-34
Querying an XMLType Table With XMLQuery and OBJECT_VALUE 8-35
Using XMLTable with PASSING and COLUMNS Clauses 8-36
Querying an XMLType Table by Using XMLTable 8-37
Querying an XMLType Column by Using XMLQuery 8-38
Querying XMLType Data by Using XMLEXISTS 8-39
Using the XMLCAST Function 8-41
Using XMLCAST and XMLQuery: Example 8-43
Querying an XML Document in the Oracle XML DB Repository 8-44
Using doc to Query Repository Documents 8-45
Using collection to Query Repository Documents In a Folder 8-46
Using XQuery to Query Sources Beyond the Database 8-47
Using a Namespace with XQuery: Examples 8-48
XQuery Support in SQL*Plus: XQUERY Command 8-50
Quiz 8-51
Summary 8-53
Practice 8-1: Overview 8-54
Practice 8-2: Overview 8-55

9 Updating XML Content Using XQuery Update

Objectives 9-2
XQuery Update 9-3
Migrating from Oracle Functions for Updating XML Data to XQuery Update 9-4
XQuery Update Snapshots 9-5
XQuery Update Snapshots: Example 9-6
Updating XML Data 9-7
Updating an Entire XML Document 9-8
General Syntax for an XQuery Update 9-9

Example: Updating XMLType Data by Using SQL UPDATE (Current State)	9-10
Example: Updating XMLType Data Using SQL UPDATE (Updated State)	9-11
Replacing XML Node Values (Current State)	9-12
Replacing XML Node Values (Updated State)	9-13
Example: Updating Multiple Text and Attribute Nodes (Current State)	9-14
Example: Updating Multiple Text and Attribute Nodes (Update Operation)	9-15
Example: Updating Multiple Text and Attribute Nodes (Result)	9-16
Example: Updating Selected Nodes Within a Collection (Current State)	9-17
Example: Updating Selected Nodes Within a Collection (Update Operation)	9-18
Example: Updating Selected Nodes Within a Collection (Result)	9-19
Updating XML Data to NULL Values Considerations	9-20
Example: Updating XML Data to NULL Values (Current State)	9-21
Example: Updating XML Data to NULL Values (Update Operation)	9-22
Example: Updating XML Data to NULL Values (Result)	9-23
Inserting Child XML Nodes	9-24
Example: Inserting an Element into a Collection (Current State)	9-25
Example: Inserting an Element into a Collection (Insert Operation)	9-26
Example: Inserting an Element into a Collection (Result)	9-27
Example: Inserting an Element Before an Element (Current State)	9-28
Example: Inserting an Element Before an Element (Result)	9-29
Example: Inserting an Element as the Last Child Element (Current State)	9-30
Example: Inserting an Element as the Last Child Element (Update and Result)	9-31
Example: Deleting XML Nodes (Current State)	9-32
Example: Deleting XML Nodes (Result)	9-33
Example: Creating XML Views of Modified XML Data	9-34
Example: Creating XML Views of Modified XML Data (Result)	9-35
Using XQuery Update to Transform Data	9-36
Quiz	9-37
Summary	9-38
Practice 9-1: Overview	9-39
Practice 9-2: Overview	9-40

10 Searching XML Content Using XQuery Full-Text

Objectives	10-2
XQuery Full-Text Search Capabilities	10-3
Combining Oracle Text Features With Oracle XML DB	10-4
Comparison of Full-Text Search and Other Search Types	10-5
Indexing for XQuery Full Text	10-6
Available Documentation	10-7
The CTX_DDL Package	10-8
The CTX_DDL Package Procedures Used in This Lesson	10-9

Oracle Text Concepts: Oracle Text Section Searching 10-10
XML Full Text Index 10-11
XML Full Text: Example 10-12
Enabling Oracle Text Section Searching: Overview 10-13
Full-Text contains Expression 10-14
Requirements for Creating XQuery Full Text Index 10-15
Indexing for XQuery Full Text: Best Performance 10-16
Using a Full-Text Index: Syntax and Example 10-17
Example: Creating an XML Full-Text Index 10-18
Example: XQuery Full Text Query 10-19
Oracle Text Reference Documentation Guide Related Topics 10-20
Using XML Schema-Based Data with XQuery Full Text 10-21
Error ORA-18177: Using XML Schema-Based Data with XQuery Full Text 10-22
Using XQuery Pragma ora:no_schema with XML Schema-Based Data 10-23
Example: Using XQuery Pragma ora:no_schema 10-24
Pragma ora:use_xmltext_idx: Forcing an XML Full-Text Index 10-25
Migrating from Oracle Text Index to XML Full-Text Index for XML 10-26
Example: Migrating from Oracle Text Index to XML Full Text Index for XML 10-27
Restrictions on Using XQuery Full Text with XMLExists 10-28
Quiz 10-29
Summary 10-30

11 Indexing XMLType Data

Objectives 11-2
Lesson Agenda 11-3
Overview of Indexing XMLType Data 11-4
Indexing XMLType Data 11-5
Lesson Agenda 11-6
XMLIndex: Overview 11-7
Benefits of Using XMLIndex Index 11-8
Structured and Unstructured Components of XMLIndex 11-10
XML Use Cases and XML Indexing 11-11
Lesson Agenda 11-12
Logical Parts of the Unstructured Component of an XMLIndex 11-13
Using XMLIndex with an Unstructured Component 11-16
Creating an XMLIndex Index Unstructured Component 11-17
Creating Secondary Indexes on the XMLIndex Unstructured Component 11-18
Specifying Storage Options When Creating an XMLIndex Index 11-19
Creating an XMLIndex Index Unstructured Component 11-20
Dictionary Views for XMLIndex 11-21
Determining the Names of the Secondary Indexes of an XMLIndex 11-22

Determining XMLIndex Index Usage	11-23
XMLIndex Path Subsetting	11-25
Specifying Paths for XMLIndex	11-26
Specifying Paths for XMLIndex: Examples	11-27
XMLIndex Parallelism	11-28
Asynchronous Maintenance of XMLIndex Index	11-29
Lesson Agenda	11-30
XMLIndex Structured Component	11-31
Creating an XMLIndex Index with Structured Component	11-32
Using A Structured XMLIndex Component for Query	11-33
Using XMLIndex: Guidelines	11-34
Turning Off XMLIndex	11-35
Summary	11-36
Practice 11: Overview	11-37

12 Generating XML Data

Objectives	12-2
Lesson Agenda	12-3
Using XQuery to Generate (Construct) XML Data from the Relational Data	12-4
List of the SQL/XML Functions	12-5
Generating an XML Document from Relational Data: Example	12-6
Lesson Agenda	12-7
XMLELEMENT Function	12-8
XMLATTRIBUTES Function	12-9
XMLFOREST Function	12-10
Generating Nested XML Elements	12-11
Using the XMLCONCAT Function	12-12
Using the XMLAGG Function	12-13
Generating Master-Detail Content	12-14
A Complex Master-Detail: Example	12-15
XMLSERIALIZE Function	12-16
The XMLCOMMENT Function	12-17
The XMLPI Function	12-18
The XMLPARSE Function	12-19
Lesson Agenda	12-20
XMLCOLATTVAL Function	12-21
XMLCDATA Function	12-22
XMLROOT Function	12-23
Lesson Agenda	12-24
DBMS_XMLGEN PL/SQL Package	12-25
Using the DBMS_XMLGEN PL/SQL Package: Steps	12-26

Generating Simple XML: Example 12-27
Generating Recursive XML with a Hierarchical Query 12-29
Quiz 12-31
Summary 12-32
Practice 12: Overview 12-33

13 Transforming and Validating XMLType Data

Objectives 13-2
Lesson Agenda 13-3
XSL Transformation and Oracle XML DB 13-4
Transformation Functions 13-5
Using the XMLTransform() Function Example: Overview 13-6
Using the XMLTransform() Function, Step 1: Create the XSL Stylesheet Table 13-7
Using the XMLTransform() Function, Step 1: Insert the XSL Stylesheet
into the Table 13-8
Using the XMLTransform() Function: Step 2, Create the XML
Documents Table 13-9
Using the XMLTransform() Function: Step 2, Insert the XML Document 13-10
Using the XMLTransform() Function, Step 3: Use XMLType columns in
XMLTransform() 13-11
Using the XMLTransform() Function 13-12
Benefits of XML Transformation 13-13
Lesson Agenda 13-14
XMLType Views 13-15
Creating an XMLType View 13-16
Lesson Agenda 13-19
Validating XMLType Instances 13-20
Performing Full Validation 13-21
Quiz 13-22
Summary 13-23
Practice 13: Overview 13-24

14 Working With the Oracle XML DB Repository

Objectives 14-2
Lesson Agenda 14-3
Oracle XML DB Repository: Overview 14-4
Oracle XML DB Repository: Architecture 14-5
Hierarchical Structures in the Repository 14-6
Links in Oracle XML DB 14-7
Oracle XML DB Repository Services 14-9
Lesson Agenda 14-11

Oracle XML DB Resource API for PL/SQL (DBMS_XDB_CONFIG)	14-12
Creating Folders and Resources by Using PL/SQL	14-13
Viewing Folders and Resources in SQL Developer	14-15
Lesson Agenda	14-16
Accessing Resources	14-17
Accessing Resources by Using SQL Access	14-18
Accessing Resources by Using SQL Access: RESOURCE_VIEW and PATH_VIEW	
SQL Functions	14-20
Determining Paths Under a Path by Using RESOURCE_VIEW: Absolute	14-21
Determining Paths Under a Path by Using RESOURCE_VIEW: Relative	14-22
Extracting Resource Metadata by Using RESOURCE_VIEW	14-23
Extracting Link and Resource Information by Using PATH_VIEW	14-24
Lesson Agenda	14-25
Navigational Access	14-26
Internet Access	14-27
Starting an HTTP Session	14-28
HTTPS Support	14-29
Starting a WebDAV Session	14-30
Quiz	14-31
Practice 14-1: Overview	14-32
Lesson Agenda	14-33
Access Control Lists	14-34
ACL-Based Security Management: Managing an ACL on a Resource	14-36
ACL-Based Security Management: Managing Privileges	14-38
Compound Documents	14-40
Compound Documents: Example	14-41
Repository Events	14-42
Implementing Repository Events	14-43
Summary	14-44
Practice 14-2: Overview	14-45

15 Using Native Oracle XML DB Web Services

Objectives	15-2
What Is a Web Service?	15-3
Web Service Standards	15-4
Web Service Architecture	15-5
Oracle XML DB Web Service: Overview	15-6
Why Native Oracle XML DB Web Services?	15-7
Steps Required to Use Web Services with Oracle XML DB	15-8
Adding a Web Services Configuration Servlet	15-9
Verifying the Addition of a Web Services Configuration Servlet	15-10

Granting Access to Web Services	15-11
Viewing the WSDL Using HTTP	15-12
Using Native Oracle XML DB Web Services	15-13
Querying Oracle XML DB By Using a Web Service	15-14
Accessing PL/SQL-Stored Procedures Using a Web Service	15-18
Defining a PL/SQL Function for Accessing the Web Service	15-19
Viewing the PL/SQL Function's WSDL	15-20
Quiz	15-21
Summary	15-23
Practice 15 Overview: Creating Web Services for Oracle XML DB	15-24

16 Exporting and Importing XML Data

Objectives	16-2
Lesson Agenda	16-3
SQL*Loader: Overview	16-4
Loading XMLType Data by Using SQL*Loader	16-6
Loading XMLType Data Stored in a Control File	16-7
Loading XMLType Data Stored in a Separate File	16-9
Lesson Agenda	16-10
Exporting and Importing XMLType Tables and Columns	16-11
Oracle Data Pump: Components	16-12
Exporting XML Schema-Based XMLType Tables	16-13
Export and Import Modes	16-14
Exporting XMLType Data: Examples	16-15
Importing XMLType Data: Example	16-16
Quiz	16-17
Summary	16-18
Practice 16: Overview	16-19

17 Workshop

Objectives	17-2
Prerequisites	17-3
Workshop I	17-4
Workshop II	17-7
Summary	17-10

18 Case Study

Objectives	18-2
Prerequisites	18-4
Case Study Username and Password Account Information	18-5
Performing XML DB Tasks as a Developer	18-7

Case Study Setup Files: DatabaseTrack Folder	18-8
Case Study Setup Files: sampleData Folder	18-9
DATA_STAGING_HOL XMLType Table	18-11
purchaseorder XMLType Table	18-12
Case Study Practices: Overview	18-13
Case Study Script Locations: lab and soln Folders	18-16
Additional Information in This Course	18-17
Oracle Database 12c Release 1 (12.1) Documentation	18-20
Summary	18-42

A Table Descriptions

Schema Descriptions	A-2
Human Resources (HR)	A-3
The HR Entity Relationship Diagram	A-4
Human Resources (HR) Row Counts	A-5
Order Entry (OE)	A-6
Order Entry (OE) Row Counts	A-8

B Using SQL Developer

Objectives	B-2
What Is Oracle SQL Developer?	B-3
Specifications of SQL Developer	B-4
SQL Developer 3.2 Interface	B-5
Creating a Database Connection	B-7
Browsing Database Objects	B-10
Displaying the Table Structure	B-11
Browsing Files	B-12
Creating a Schema Object	B-13
Creating a New Table: Example	B-14
Using the SQL Worksheet	B-15
Executing SQL Statements	B-19
Saving SQL Scripts	B-20
Executing Saved Script Files: Method 1	B-21
Executing Saved Script Files: Method 2	B-22
Formatting the SQL Code	B-23
Using Snippets	B-24
Using Snippets: Example	B-25
Using Recycle Bin	B-26
Debugging Procedures and Functions	B-27
Database Reporting	B-28
Creating a User-Defined Report	B-29

Search Engines and External Tools	B-30
Setting Preferences	B-31
Resetting the SQL Developer Layout	B-33
Data Modeler in SQL Developer	B-34
Summary	B-35

C Using JDeveloper

Objectives	C-2
Oracle JDeveloper	C-3
Database Navigator	C-4
Creating a Database Connection	C-5
Browsing Database Objects	C-6
Executing SQL Statements	C-7
Creating Program Units	C-8
Compiling	C-9
Running a Program Unit	C-10
Dropping a Program Unit	C-11
Structure Window	C-12
Editor Window	C-13
Application Navigator	C-14
Deploying Java Stored Procedures	C-15
Publishing Java to PL/SQL	C-16
External Tools with JDeveloper	C-17
Managing External Tools	C-18
Creating a New External Tool: Step 1 of 11 – Type	C-19
Creating a New External Tool: Step 2 of 11 – External Program Options	C-20
Creating a New External Tool: Step 9 of 11 – Display	C-21
Creating a New External Tool: Step 10 of 11 – Integration	C-22
Creating a New External Tool: Step 11 of 11 – Availability	C-23
Using the Newly Added “gedit Text Editor” External Tool	C-24
Editing an External Tool	C-25
Deleting an External Tool	C-26
How Can I Learn More About JDeveloper 11g ?	C-27
Summary	C-28

D PL/SQL API for XMLType

Objectives	D-2
Lesson Agenda	D-3
PL/SQL API Packages for XMLType	D-4
DBMS_XMLSTORE PL/SQL Package	D-5
Using the DBMS_XMLSTORE PL/SQL Package: Steps	D-6

Inserting with DBMS_XMLSTORE	D-8
Updating with DBMS_XMLSTORE	D-11
Deleting with DBMS_XMLSTORE	D-13
Lesson Agenda	D-15
Document Object Model of XML	D-16
W3C Document Object Model Recommendation: Overview	D-17
Processing XML Structures with DBMS_XMLDOM	D-18
DOM Node Types: Inheritance	D-19
Using DBMS_XMLDOM to Create DOM Document Handles	D-20
Creating a DOM Document	D-21
Accessing and Manipulating a DOM Document	D-22
Adding Nodes to a DOM Document	D-24
Lesson Agenda	D-28
XML Parsers	D-29
PL/SQL Parser API for XMLType (DBMS_XMLPARSER)	D-31
Obtaining a DOM Document Interface by Using DBMS_XMLPARSER	D-32
Parsing an XML Document Without DTD Reference by Using DBMS_XMLPARSER	D-33
Parsing an XML Document with DTD Reference by Using DBMS_XMLPARSER	D-34
Lesson Agenda	D-37
PL/SQL XSLT Processor for XMLType (DBMS_XSLPROCESSOR)	D-38
Transforming an XML Document by Using DBMS_XSLPROCESSOR	D-39
Using DBMS_XSLPROCESSOR to Transform an XML Document: Example	D-40
DBMS_XSLPROCESSOR Subprograms	D-43
DBMS_XSLPROCESSOR.PROCESSXSL()	D-44
Summary	D-45

E XML Review

Objectives	E-2
Extensible Markup Language	E-3
Example: A Simple XML Document	E-4
Markup Rules for Elements	E-5
XML Attributes	E-7
XML Character Data (CDATA)	E-9
Well-Formed XML Documents	E-10
Document Type Definition (DTD)	E-11
Why Validate an XML Document?	E-12
XML Namespace	E-13
Why Use XML Namespaces?	E-14
Declaring XML Namespaces	E-15

XML Namespace Prefixes	E-16
XML Namespace Declarations: Example	E-17
XML Schema	E-18
Benefits of XML Schemas	E-19
XML Schema Document: Example	E-20
Validating an XML Document with an XML Schema Document	E-21
Referencing an XML Schema with the schemaLocation Attribute	E-22
Components of an XML Schema	E-23
XML Schema Components: Example	E-24
<schema> Declaration	E-25
Declaring an Element	E-28
Built-In XML Schema Data Types	E-30
Declaring a <simpleType> Component	E-32
Declaring <complexType> Components	E-34
Declaring a <sequence>	E-35
Declaring a <choice>	E-36
Declaring an Empty Element	E-37
Declaring Attributes	E-38
Attribute Declarations: Example	E-40
Validating an XML Document with its XML Schema by Using oraxml	E-41
XML Path Language	E-42
XPath Model	E-43
XPath Expressions	E-45
Location Path Expression	E-46
Location Path Expression: Example	E-47
Results of Location Path Expressions	E-48
Location Steps in XPath Expressions	E-49
XPath Axes	E-51
XPath Node Test Types	E-53
Unabbreviated and Abbreviated Expressions	E-54
Abbreviated XPath Expression: Examples	E-56
XPath Predicates	E-58
XPath Functions	E-61
Boolean Functions	E-62
Node-Set Functions	E-63
String Functions	E-65
XSLT and XPath	E-67
XML Stylesheet Language (XSL) Transformations	E-69
XSLT Style Sheet	E-71
XSLT Style Sheet: Example	E-72
Using an XSLT Style Sheet with an XML Document	E-73

- Viewing the Transformed Document E-74
- XQuery E-75
- XQuery Terminology and Syntax Rules E-76
- XQuery Expressions E-77
- Primary Expressions: Variables and Literals E-78
- Sequence Expressions E-80
- Path Expressions E-81
- Using Conditional Expressions E-83
- Summary E-84

F Glossary

Preface

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Profile

Before You Begin This Course

Before you begin this course, you should have experience in XML, Document Type Definitions (DTDs), Namespaces, XML schemas, XML Path Language (XPath), Extensible Stylesheet Language (XSL), and XQuery. The following Oracle University courses are a prerequisite:

- *Oracle XML Fundamentals*
 - *Oracle Database: Program with PL/SQL*
- The following is a list of the suggested prerequisites:
- *Familiarity with Oracle SQL Developer and SQL*Plus*

How This Course Is Organized

Oracle Database 12c: Use XML DB is an instructor-led course featuring lectures and hands-on exercises. Online demonstrations, hands-on practices, workshops, and a case study reinforce the concepts and skills that are introduced.

Related Publications

Oracle Publications

Title	Part Number
<i>Oracle XML DB Developer's Guide 12c Release 1</i>	E17603-09
<i>Oracle Database SQL Language Reference 12c Release 1</i>	E17209-14
<i>Oracle Database PL/SQL Language Reference 12c Release 1</i>	E17622-18
<i>Oracle Database PL/SQL Packages and Types Reference 12c Release 1</i>	E17602-14
<i>Oracle SQL Developer User's Guide Release 3.2</i>	E35117-05
<i>Oracle Database Object-Relational Developer's Guide 12c Release 1</i>	E16801-07
<i>Oracle Database SQLJ Developer's Guide 12c Release 1</i>	E17660-11
<i>Oracle Text Reference 12c Release 1 (12.1)</i>	E17747-07
<i>Oracle Text Application Developer's Guide 12c Release 1</i>	E17748-07

Additional Publications

- System release bulletins
- Installation and user's guides
- *read.me* files
- International Oracle User's Group (IOUG) articles
- *Oracle Magazine*

Typographic Conventions

The following two lists explain Oracle University typographical conventions for words that appear within regular text or within code samples.

1. Typographic Conventions for Words Within Regular Text

Convention	Object or Term	Example
Courier New	User input; commands; column, table, and schema names; functions; PL/SQL objects; paths	Use the SELECT command to view information stored in the LAST_NAME column of the EMPLOYEES table. Enter 300. Log in as scott
Initial cap	Triggers; user interface object names, such as button names	Assign a When-Validate-Item trigger to the ORD block. Click the Cancel button.
Italic	Titles of courses and manuals; emphasized words or phrases; placeholders or variables	For more information on the subject see <i>Oracle SQL Reference Manual</i> Do <i>not</i> save changes to the database. Enter <i>hostname</i> , where <i>hostname</i> is the host on which the password is to be changed.
Quotation marks	Lesson or module titles referenced within a course	This subject is covered in Lesson 3, “Working with Objects.”

Typographic Conventions (continued)

2. Typographic Conventions for Words Within Code Samples

Convention	Object or Term	Example
Uppercase	Commands, functions	SELECT employee_id FROM employees;
Lowercase, italic	Syntax variables	CREATE ROLE <i>role</i> ;
Initial cap	Forms triggers	Form module: ORD Trigger level: S_ITEM.QUANTITY item Trigger name: When-Validate-Item . . .
Lowercase	Column names, table names, filenames, PL/SQL objects	. . . OG_ACTIVATE_LAYER (OG_GET_LAYER ('prod_pie_layer')) . . . SELECT last_name FROM employees;
Bold	Text that must be entered by a user	CREATE USER scott IDENTIFIED BY tiger ;

1

Introduction



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Discuss the course objectives, prerequisites and suggested prerequisites, students' introductions, and agenda
- List the database and XML schemas used in this course
- List the appendices used in this course
- Identify the relevant documentation and other resources



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Course objectives and course agenda
- The database and XML schemas, appendices, and SQL environments used in this course
- Overview of Oracle SQL Developer
- Oracle 12c documentation and additional resources

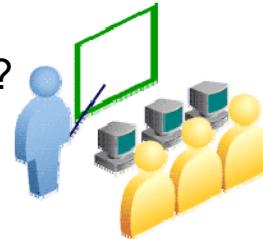


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Questions About You

To ensure that the class can be customized to meet your specific needs and to encourage interaction among all, answer the following questions:

- Which organization do you work for?
- What is your role in your organization?
- What is your level of XML expertise?
- Do you have the course prerequisites?
- What Oracle Database versions do you use?
- How do you plan to use Oracle XML DB?
- What do you hope to receive from this course?



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can benefit most from this class if you know the backgrounds of your classmates and the issues that they face in the development and management of a data warehouse. Each student has unique perspectives, experience, and knowledge from which you all can learn.

Course Objectives

After completing this course, you should be able to:

- Explain the key features of Oracle XML DB
- Manage XML storage in Oracle XML DB
- Manage changes in an XML schema
- Partition XMLType tables
- Retrieve XML data in Oracle XML DB
- Create and use indexes on XML data
- Generate, manipulate, and transform XML in Oracle XML DB
- Use the Oracle XML DB Repository
- Use Native Oracle XML DB Web Services
- Import and export XML data



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Course Prerequisites

The following Oracle University courses are prerequisites for this course:

- *Oracle XML Fundamentals*
- *Oracle Database: Program with PL/SQL*



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Suggested Course Agenda

Day 1:

- Lesson 1: Introduction
- Lesson 2: XML Basic Review
- Lesson 3: Introduction to Oracle XML DB
- Lesson 4: Storing XML Data in Oracle XML DB
- Lesson 5: Using XML Schema with Oracle XML DB

Day 2:

- Lesson 6: Oracle XML DB Manageability
- Lesson 7: Partitioning XMLType Tables
- Lesson 8: Using XQuery to Retrieve XML Data in Oracle XML DB
- Lesson 9: Updating XML Content Using XQuery Update



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This agenda is a suggested list of lessons to be covered on each day of the three-day course.

Day 1

- **Lesson 1, Introduction (this lesson):** Introduces the students to the course information such as the course objectives, prerequisites and suggested prerequisites, students' introductions, suggested agenda, and so on
- **Lesson 2, XML Basic Review:** Provides a quick review to XML to gauge the students' knowledge of the basic XML skills required and used in this course
- **Lesson 3, Introduction to Oracle XML:** Provides a high-level review of Oracle XML DB, its features, and benefits.
- **Lesson 4, Storing XML Data in Oracle XML DB:** Focuses on creating XMLType objects, identifying the storage options available, and loading data into XMLType objects
- **Lesson 5, Using XML Schema With Oracle XML DB:** Covers creating XML schema-based XMLType tables and columns. This lesson also discusses specifying binary XML storage of XML schema-based data. Students also learn how to perform XML schema registration and evolution.

Day 2

- **Lesson 6, Oracle XML DB Manageability:** Explains how to reduce the time and cost associated with annotating XML schemas.
- **Lesson 7, Partitioning XMLType tables:** Explains how to partition XMLTYPE tables and columns that are stored as object relational. It also specifies partitioning information for Binary XML Tables.
- **Lesson 8, Using XQuery to Retrieve XML Data in Oracle XML DB:** Describes the functions that are used to query XMLType data stored in the database
- **Lesson 9, Updating XML Content Using XQuery Update:** Describes how to use the new support for the XQuery Update recommendation to perform fragment and node-level updates by using the W3C Query language

Day 3

- **Lesson 10, Searching XML Content Using XQuery Full-Text:** Describes how XQuery Full Text reduces the amount of development effort required to develop document-centric XML applications by supporting the W3C standard for performing full-text searches on XML content stored in the database. This feature enables text searching of XML fragments in addition to leaf-level and document-level text searches.
- **Lesson 11, Indexing XMLTYPE Data:** Describes the methods for indexing XMLType data
- **Lesson 12, Generating XML Data:** Describes the functions and procedures that are available to create Extensible Markup Language (XML) data from relational data
- **Lesson 13, Transforming and Validating XML Data:** Explains the functions that are available to modify XML data

Suggested Course Agenda

Day 3:

- Lesson 10: Searching XML Content Using XQuery Full-Text
- Lesson 11: Indexing XMLType Data
- Lesson 12: Generating XML Data
- Lesson 13: Transforming and Validating XML Data

Day 4:

- Lesson 14: Working With the Oracle XML DB Repository
- Lesson 15: Using Native Oracle XML DB Web Services



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Day 4

- **Lesson 14, Working With the Oracle XML DB Repository:** Focuses on Oracle XML DB Repository, which is a feature of XML DB that integrates content management services with Oracle Database. The second half of this lesson focuses on securing Oracle XML DB repository using ACL, describing compound documents and repository events, and managing repository events.
- **Lesson 15, Using Native Oracle XML DB Web Services:** Explains how to configure and enable Web services for Oracle XML DB. The lesson also focuses on how to query Oracle XML DB by using a web service and how to access PL/SQL stored procedures by using a web service.
- **Lesson 16, Exporting and Importing XML Data:** Explains how to load XMLType data into relational tables by using SQL*Loader. The lesson also focuses on the Oracle Data Pump technology to import and export XMLType tables for use with Oracle XML DB.

Suggested Course Agenda

Day 5:

- Lesson 16: Exporting and Importing XML Data
- Lesson 17: Workshop
- Lesson 18: Case Study



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

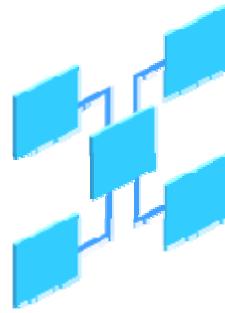
Day 5

- **Lesson 17, Workshop:** In this workshop, you perform tasks such as storing XML data, creating folders and resources, and accessing resources in the Oracle XML DB Repository. You also use SQL/XML standard functions and XQuery to retrieve XML data that is stored in Oracle XML DB. Finally, you create XMLIndex index to improve the performance of XQuery expressions.
- **Lesson 18, Case Study:** In this case study, you apply what you learned in this course. You use XQuery and SQL/XML functions to query XML documents stored in the PURCHASEORDER table in the XDBOE Schema. You also index binary XML content stored in the Oracle database to improve performance. You update XML documents using XQuery update and create and use XML aware full-text index. You use an XML schema to optimize storage, index, query, and manage structured XML data. You use the XQuery expressions with the Binary XML table on the Object-Relational table. You also create a partitioned object-relational XMLType table. Finally, you create a persistent XML view of the relational data.

Database Schemas Used in This Course

The following two database schemas are used in the course:

- Human Resources (HR) schema
- Order Entry (OE) schema



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Database Schemas

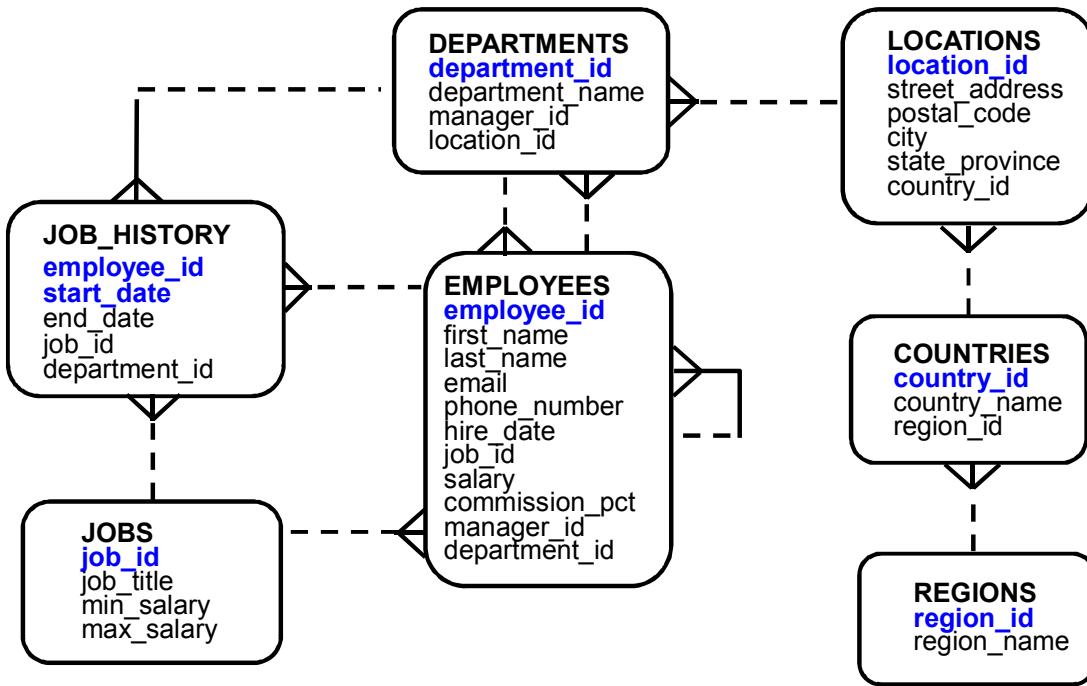
The fictitious company in the sample schemas operates worldwide to fulfill orders for several products. The company has several divisions, with each division represented by a schema:

- The Human Resources division tracks information about the company's employees and facilities. All the scripts that are necessary to create the `HR` schema are in the `$ORACLE_HOME/demo/schema/human_resources` folder.
- The Order Entry division tracks product inventories and sales of the company's products through various channels. All the scripts that are necessary to create the `OE` schema are in the `$ORACLE_HOME/demo/schema/order_entry` folder.

Note

- “**Appendix A: Table Descriptions**” contains more information about the sample schemas.
- The code examples and the practices in this course specify the schema that must be used.

The Human Resources (hr) Schema



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

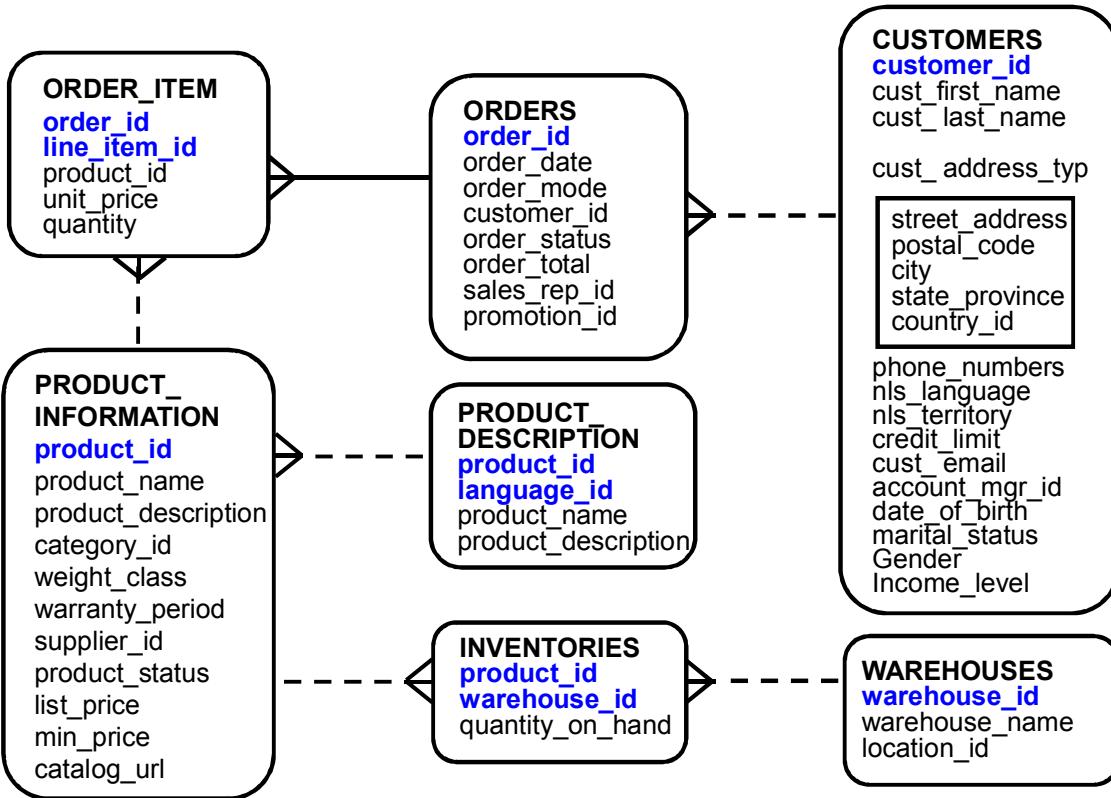
In the HR records, each employee has an identification number, email address, job identification code, salary, and manager. Some employees earn commissions in addition to their salary.

The company also tracks information about jobs within the organization. Each job has an identification code, job title, and a minimum and maximum salary range for the job. Some employees have been with the company for a long time and have held different positions within the company. When an employee resigns, the duration for which the employee worked working, the job identification number, and the department are recorded.

The sample company is regionally diverse, so it tracks the locations of its warehouses and departments. Each employee is assigned to a department and each department is identified by a unique department number or a short name. Each department is associated with one location and each location has a full address that includes the street name, postal code, city, state or province, and the country code.

In places where the departments and warehouses are located, the company records details such as the country name, currency symbol, currency name, and the region where the country is located geographically.

The Order Entry (oe) Schema



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The company sells several categories of products, including computer hardware and software, music, clothing, and tools. The company maintains product information that includes product identification numbers, the category into which the product falls, the weight group (for shipping purposes), the warranty period if applicable, the supplier, the status of the product, a list price, a minimum price at which a product is sold, and a URL for manufacturer information.

The Purchase Order XML Schema, purchaseorder.xsd, Used in This Course

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
  <xs:element name="PurchaseOrder" type="PurchaseOrderType"/>
  <xs:complexType name="PurchaseOrderType">
    <xs:sequence>
      <xs:element name="Reference" type="ReferenceType"/>
      <xs:element name="Actions" type="ActionsType"/>
      <xs:element name="Reject" type="RejectionType" minOccurs="0"/>
      <xs:element name="Requestor" type="RequestorType"/>
      <xs:element name="User" type="UserType"/>
      <xs:element name="CostCenter" type="CostCenterType"/>
      <xs:element name="ShippingInstructions" type="ShippingInstructionsType"/>
      <xs:element name="SpecialInstructions" type="SpecialInstructionsType"/>
      <xs:element name="LineItems" type="LineItemsType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="LineItemsType">
    <xs:sequence>
      <xs:element name="LineItem" type="LineItemType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="LineItemType">
    <xs:sequence>
```

....



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example shows the partial Purchase Order XML schema that is used in this course as an XML file, `purchaseOrder.xsd`.

The purchase order XML schema demonstrates some key features of a typical XML document:

- Global element `PurchaseOrder` is an instance of the `complexType PurchaseOrderType`.
- `PurchaseOrderType` defines the set of nodes that make up a `PurchaseOrder` element.
- `LineItems` element consists of a collection of `LineItem` elements.
- Each `LineItem` element consists of two elements: `Description` and `Part`.
- `Part` element has the following attributes: `ID`, `Quantity`, and `UnitPrice`.

For the complete code listing and graphical representation of the Purchase Order XML schema, see Chapter 3, *Using Oracle XML DB*, or Appendix A, *Oracle-Supplied XML Schemas and Examples* in the Oracle XML DB Developer's Guide 12c Release 1 (12.1) documentation reference.

Appendices Used in This Course

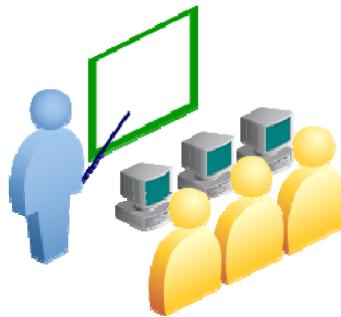
Appendix Title
<i>Appendix A: Table Descriptions</i>
<i>Appendix B: Using SQL Developer</i>
<i>Appendix C: Using JDeveloper</i>
<i>Appendix D: PL/SQL API for XMLType</i>
<i>Appendix E: XML Review</i>
<i>Appendix F: Glossary</i>



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Class Account Information

- Your account IDs are `hr` and `oe`.
- The password matches your account ID.
- For the case study, the account ID is `XDBOE` and the password is `oracle`.
- Each machine has a stand-alone installation of Oracle Database 12c Enterprise Edition Release 1 (12.1) for Linux with access to the `hr` and `oe` sample schemas.



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Course Environment

- Client software
 - Oracle SQL Developer (used in this course)
 - Oracle SQL*Plus (used as needed in this course)
 - Oracle JDeveloper IDE (optional)
- Oracle Database 12c (XML DB part of the database)
- hr and oe Oracle sample schemas with SYSDBA and XDBADMIN privilege to create and register XML schemas



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

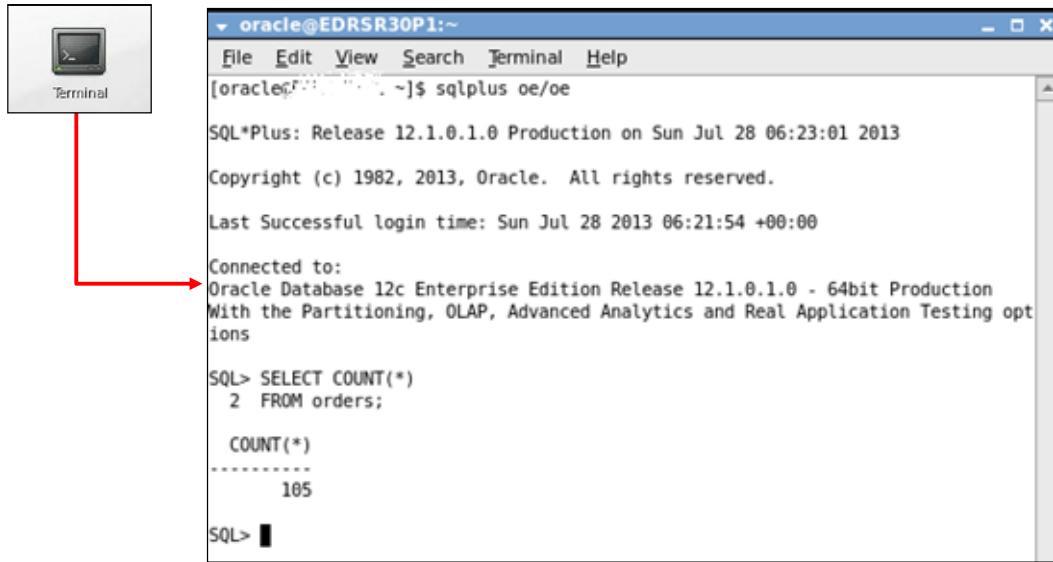
Client Software

There are many tools that provide an environment for executing SQL statements. Oracle provides several tools that can be used to execute SQL statements. Some of the tools that are available for use in this course are:

- **Oracle SQL Developer:** A graphical tool
- **Oracle SQL*Plus:** A window or command-line application
- **Oracle JDeveloper:** A Windows-based integrated development environment (IDE)
- **Oracle Database 12c** (Oracle XML DB part of the database)
- HR and OE Oracle sample schemas with SYSDBA and XDBADMIN privileges to create and register XML schemas

Note: Most of the code and screen examples presented in the course notes are from output in the SQL Developer environment; there are some instances where SQL*Plus using a terminal window was used instead of SQL Developer. Such instances are well documented.

Entering SQL Statements Using Oracle SQL*Plus in a Terminal Window

The Oracle logo, consisting of the word "ORACLE" in a bold, white, sans-serif font.

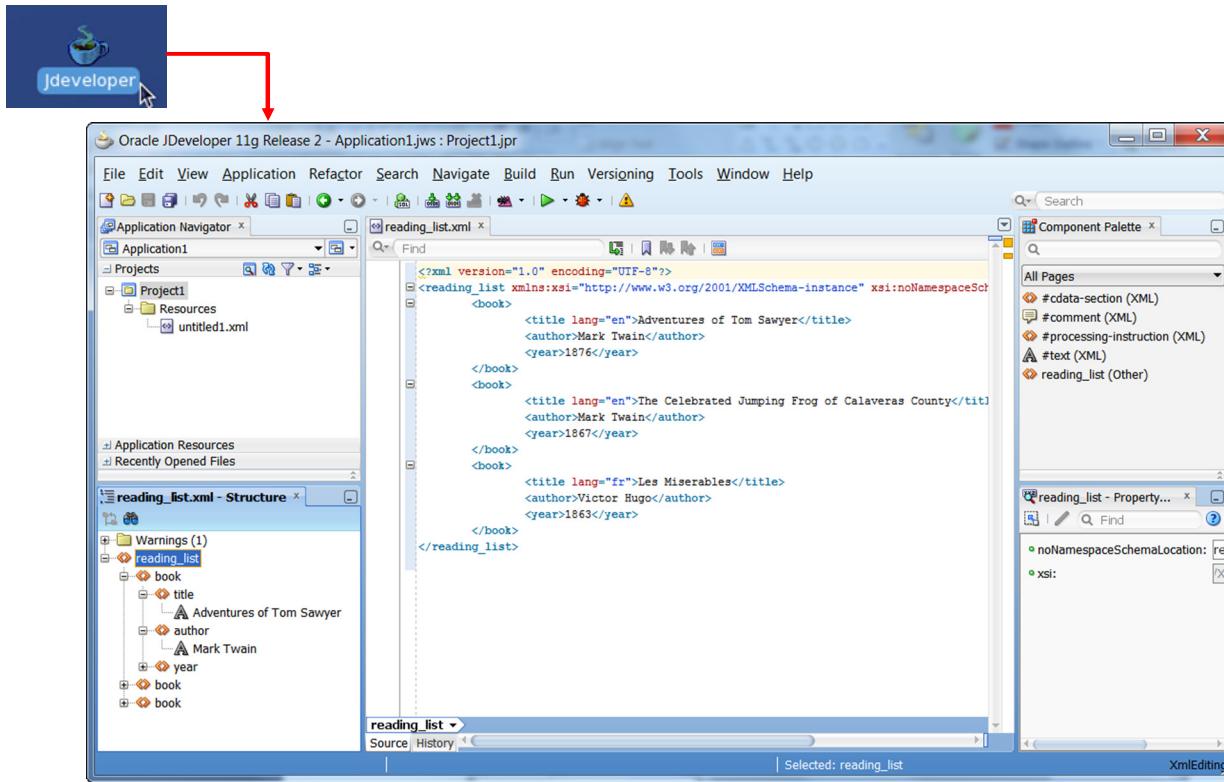
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle SQL*Plus is a command-line application with which you can submit SQL statements and PL/SQL blocks for execution and receive the results in an application or command window.

SQL*Plus is:

- Shipped with the database
- Installed on the database server system and is accessed from an icon or the command line

Using XML in Oracle JDeveloper



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

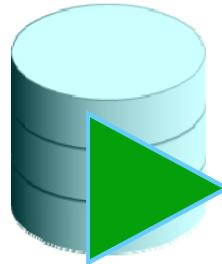
With Oracle JDeveloper, developers can create, edit, test, and debug PL/SQL code by using a sophisticated GUI. Oracle JDeveloper is a part of Oracle Developer Suite and is also available as a separate product.

To use JDeveloper with XML, you first create a database connection to enable JDeveloper to access a database schema owner for the subprograms.

Note: For more information about using JDeveloper, see Appendix C.

Entering SQL Statements and Coding PL/SQL Using Oracle SQL Developer

- Oracle SQL Developer is a free graphical tool that enhances productivity and simplifies database development tasks.
- You can connect to any target Oracle database schema by using standard Oracle database authentication.
- You use SQL Developer in this course.



SQL Developer

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle SQL Developer

Oracle SQL Developer is a free graphical tool designed to improve productivity and simplify the development of everyday database tasks. With a few clicks, you can easily create and debug stored procedures, test SQL statements, and view optimizer plans.

SQL Developer, the visual tool for database development, simplifies the following tasks:

- Browsing and managing database objects
- Executing SQL statements and scripts
- Editing and debugging PL/SQL statements
- Creating reports

You can connect to any target Oracle database schema by using standard Oracle database authentication. When connected, you can perform operations on objects in the database.

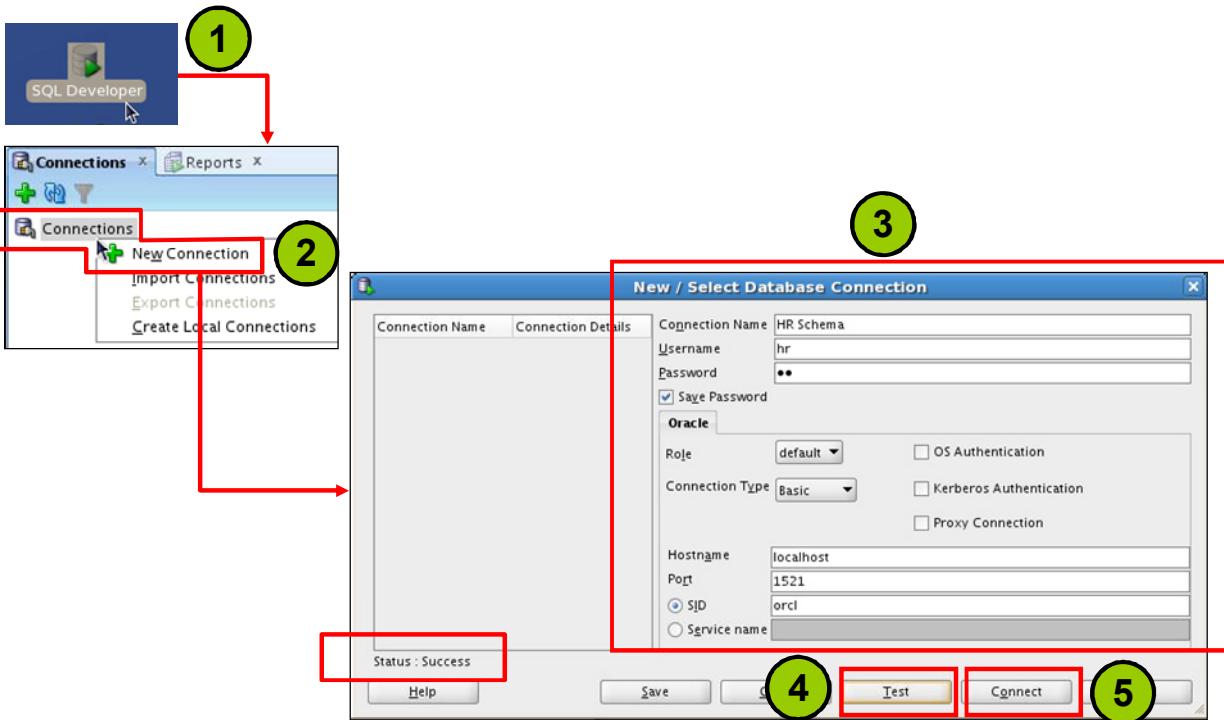
Lesson Agenda

- Course objectives and course agenda
- The schemas, appendices, and SQL environments used in this course
- Overview of Oracle SQL Developer
- Oracle 12c documentation and additional resources



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Starting Oracle SQL Developer and Creating a Database Connection



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Creating a Database Connection

To create a database connection, perform the following steps:

1. Double-click your SQL Developer icon.
2. On the Connections tab, right-click Connections, and then select New Connection.
3. Enter the connection name, username, password, host name, and system identifier (SID) for the database that you want to connect to.
4. Click Test to make sure that the connection has been set correctly.
5. Click Connect.

On the Basic tabbed page at the bottom, enter the following:

- **Hostname:** Host system for the Oracle database
- **Port:** Listener port
- **SID:** Database name
- **Service name:** Network service name for a remote database connection

If you select the Save Password check box, the password is saved to an Extensible Markup Language (XML) file. So, after you close the SQL Developer connection and open it again, you are not prompted for the password.

Creating Database Schema Objects

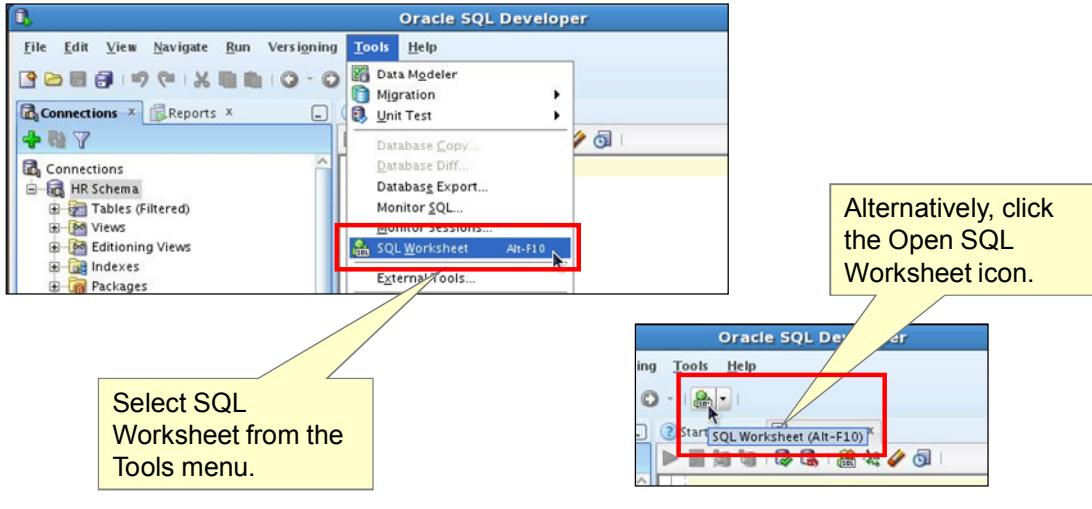
- You can create any database schema object in Oracle SQL Developer by using one of the following methods:
 - Executing a SQL statement in the SQL Worksheet
 - Using the context menu
- Edit the objects by using an edit dialog box or one of the many context-sensitive menus.
- View the data definition language (DDL) for adjustments such as creating a new object or editing an existing schema object.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Using the SQL Worksheet

- Use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements.
- Specify any actions that can be processed by the database connection associated with the worksheet.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When you connect to a database, a SQL Worksheet window for that connection is automatically opened. You can use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. The SQL Worksheet supports SQL*Plus statements to a certain extent. SQL*Plus statements that are not supported by the SQL Worksheet are ignored and not passed to the database. You can specify actions that can be processed by the database connection associated with the worksheet, such as:

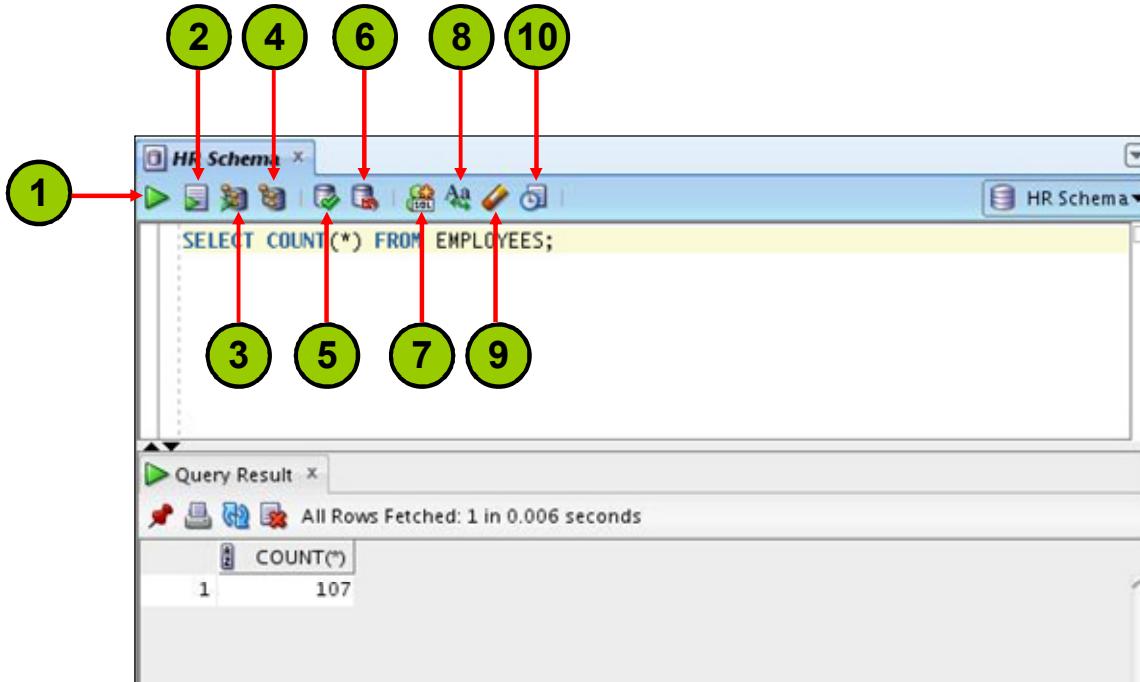
- Creating a table
- Inserting data
- Creating and editing a trigger
- Selecting data from a table
- Saving the selected data to a file

You can display a SQL Worksheet by performing one of the following steps:

- Select Tools > SQL Worksheet.
- Click the Open SQL Worksheet icon.

You can use the shortcut keys or icons to perform certain tasks such as executing a SQL statement, running a script, and viewing the history of SQL statements that you have executed.

Using the SQL Worksheet Toolbar

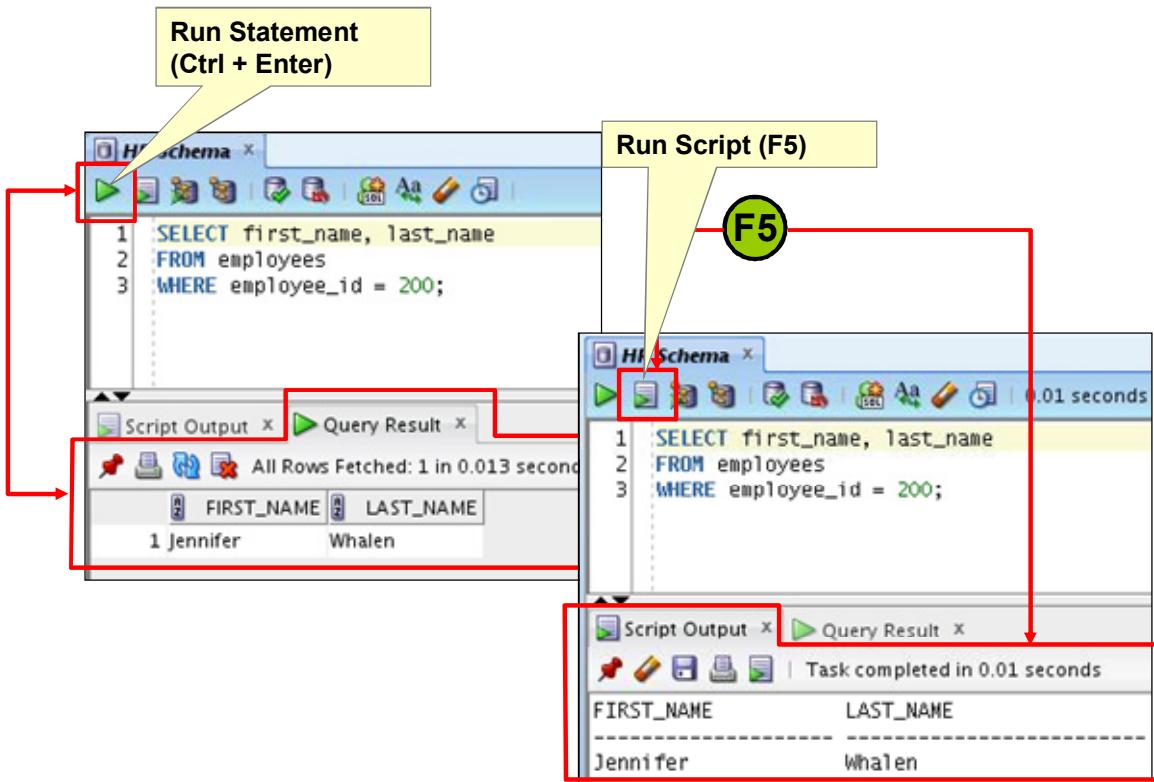


ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

1. **Execute Statement (F9):** Executes the statement at the cursor in the Enter SQL Statement box
2. **Run Script (F5):** Executes all statements in the SQL Worksheet text box using the Script Runner. You can use substitution variables in the SQL statements, but not bind variables.
3. **Autotrace (F6):** Generates trace information for the statement
4. **Explain Plan (F10):** Generates the execution plan
5. **Commit:** Writes any changes to the database and ends the transaction
6. **Rollback:** Discards any changes to the database, without writing them to the database, and ends the transaction
7. **Unshared SQL Worksheet (Ctrl + Shift + N):** Creates an unshared worksheet that uses a separate database connection from the Connections navigator
8. **To Upper/Lower/Initcap (Ctrl + quote):** Changes the letter case
9. **Clear:** Erases the statement or statements in the SQL Worksheet text box
10. **SQL History:** Displays information about SQL statements that you have executed

Executing SQL Statements



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

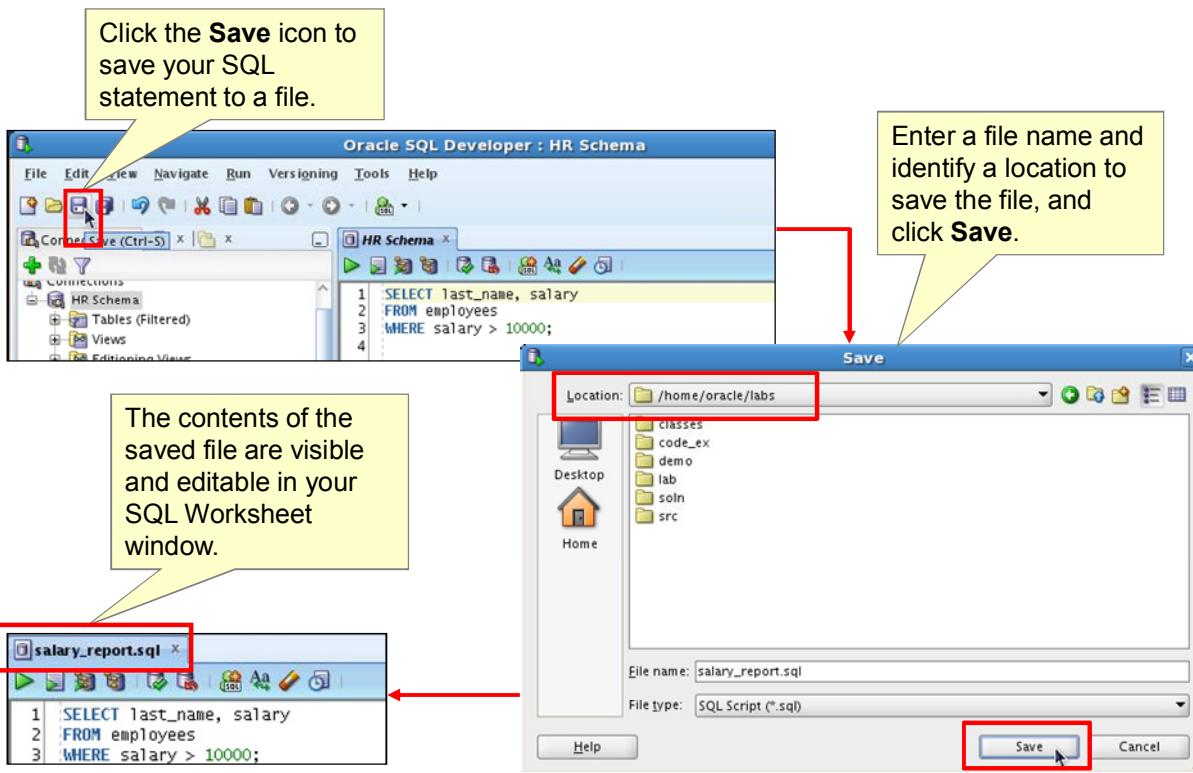
In the SQL Worksheet, you can use the Enter SQL Statement box to enter a single or multiple SQL statements. For a single statement, the semicolon at the end is optional.

When you enter a statement, the SQL keywords are automatically highlighted. To execute a SQL statement, ensure that your cursor is within the statement and click the Execute Statement icon. Alternatively, you can press Ctrl + Enter. The result is displayed in the Query Result tabbed page.

To execute multiple SQL statements and see the results, click the Run Script icon. Alternatively, you can press F5. The result is displayed in the Script Output tabbed page.

The example in the slide shows the difference in output for the same query when Ctrl + Enter or Execute Statement is used, and when F5 or Run Script is used.

Saving SQL Scripts



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

You can save your SQL statements from the SQL Worksheet into a text file. To save the contents of the Enter SQL Statement box, follow these steps:

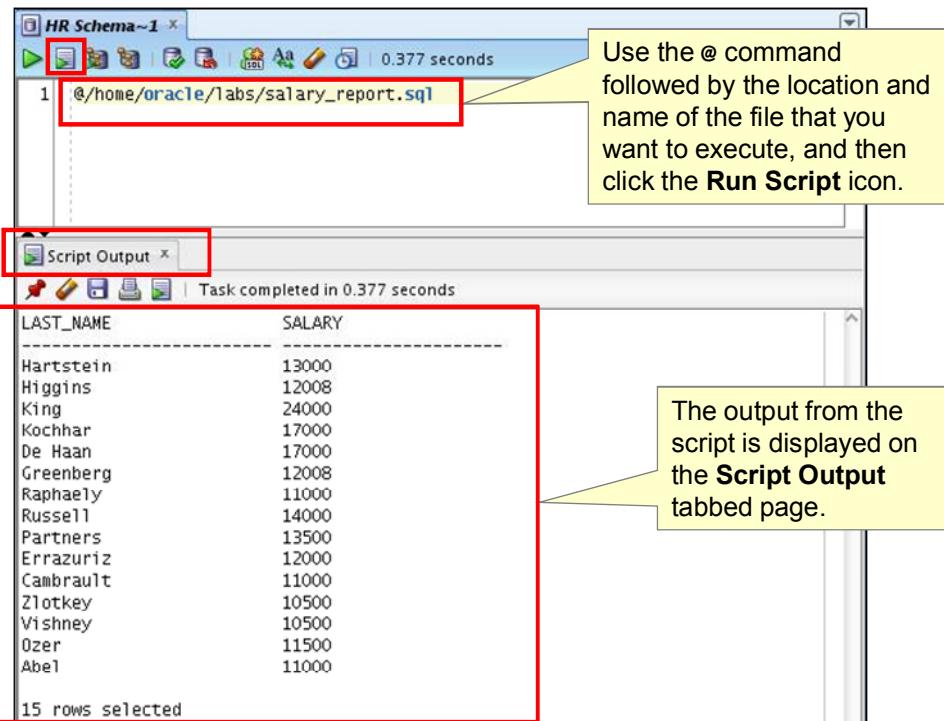
1. Click the Save icon or use the File > Save menu item.
2. In the Save dialog box, enter a file name, the location where you want the file saved, and then click Save.

After you save the contents to a file, the Enter SQL Statement window displays a tabbed page of your file contents. You can have multiple files open at the same time. Each file is displayed as a tabbed page.

Script Pathing

You can select a default path to look for scripts and to save scripts. Under Tools > Preferences > Database > Worksheet Parameters, enter a value in the “Select default path to look for scripts” field.

Executing Saved Script Files by Using the @ Command



ORACLE

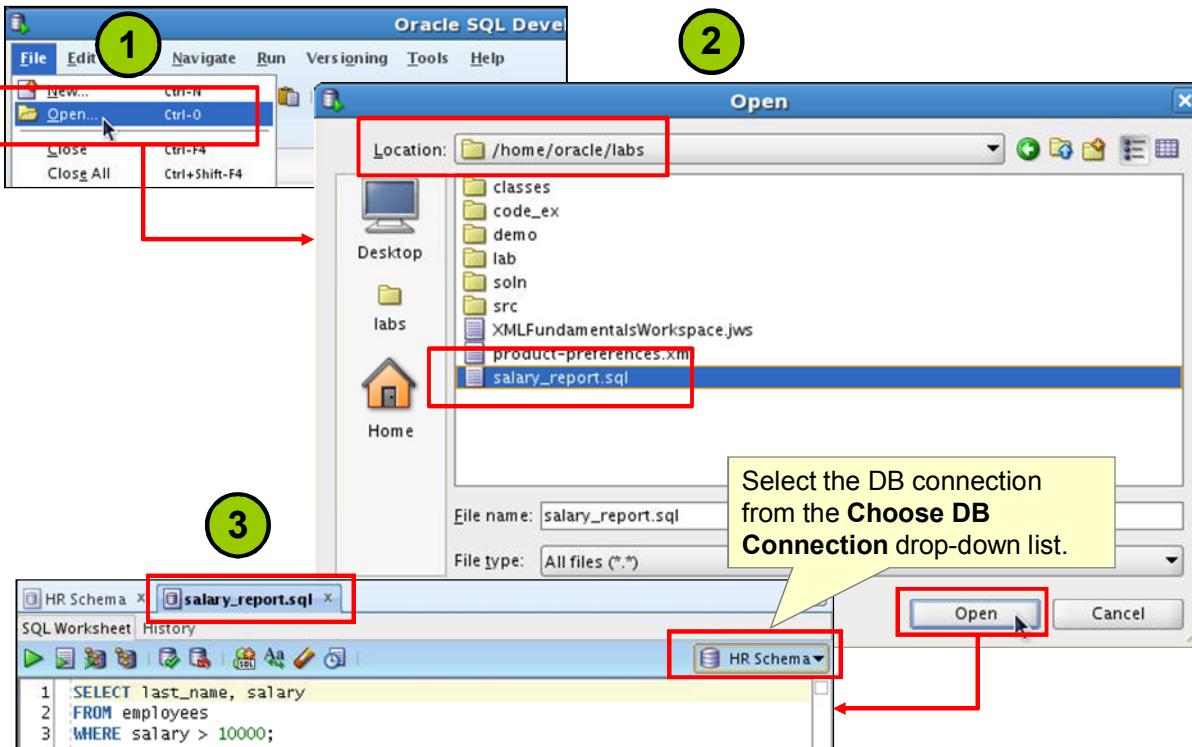
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To run a saved SQL script, perform the following:

1. Use the @ command, followed by the location, and the name of the file that you want to run, in the SQL Worksheet text box.
2. Click the Run Script icon.

The results from running the file are displayed on the Script Output tabbed page. You can also save the script output by clicking the Save icon on the Script Output tabbed page. The Save dialog box appears and you can identify a name and location for your file.

Executing SQL Scripts



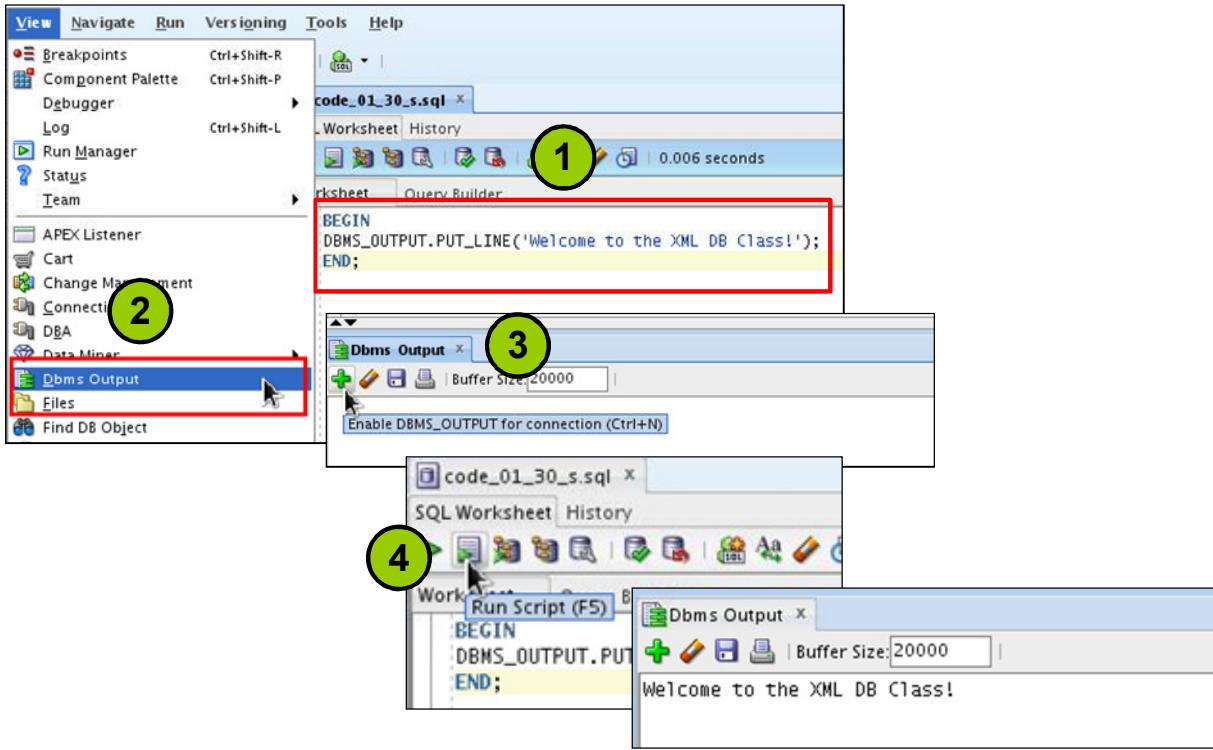
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To run the saved SQL scripts, perform the following steps:

1. Select File > Open from the File menu. The Open dialog box is displayed. You can also click the Open icon on the toolbar, or click the Files tab, and then navigate to the file.
2. In the Open dialog box, browse to the /home/oracle/labs folder or to the location where you saved the script file, select the file, and then click Open.
3. The script file opens in a new worksheet. The SQL Worksheet toolbar is disabled. Select the database connection from the Choose DB Connection drop-down list. The Worksheet toolbar is now enabled. Now, you can run the script by either clicking the Execute Statement icon or the Run Script icon. Again, ensure that you do not enter any other SQL statement in the same worksheet. To continue with other SQL queries, open a new worksheet.

Creating an Anonymous PL/SQL Block



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can create an anonymous block and display the output of the `DBMS_OUTPUT` package. To create an anonymous block and view the results, perform the following steps:

1. Enter the PL/SQL code in the SQL Worksheet area.
2. Select View > Dbms Output from the menu bar. The Dbms Output pane is displayed.
3. Click the Add new Dbms Output tab icon (the + icon) on the Dbms Output pane toolbar. The Select Connection window is displayed. Select the appropriate connection from the Connection drop-down list, and then click OK. The message Set serveroutput on is displayed.
4. Click the Execute Statement or the Run Script icons above the Enter SQL Statement box. The output is displayed in the DBMS Output pane.

Lesson Agenda

- Course objectives and course agenda
- The schemas, appendices, and SQL environments used in this course
- Overview of Oracle SQL Developer
- Oracle 12c documentation and additional resources



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Available Oracle University Courses at: education.oracle.com

The screenshot shows a web browser window for Oracle University. The URL is education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=501. The search results page displays three course categories: 'XML Fundamentals', 'Oracle 11g: XML Fundamentals', and 'Oracle Database 11g: Use XML DB NEW'. Each category has options for 'Classroom Training', 'Live Virtual Class', and 'Self-Study Course', each with a 'Register Interest' button. A red box highlights the search bar with the keyword 'Xml Db'. The left sidebar includes filters for 'Country' (United States), 'Category' (Database), and 'Course Material Language' (English). A contact link is also present.

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Where to Go for More Information?

Topic	Web Site
XML DB Home Page	http://www.oracle.com/technetwork/database-features/xmldb/overview/index.html
Oracle Technology Network (OTN)	http://www.oracle.com/us/products/database/overview/index.html
Education and Training	http://education.oracle.com
Product Documentation	http://www.oracle.com/technology/documentation
Product Downloads	http://www.oracle.com/technology/software
Product Articles	http://www.oracle.com/technology/pub/articles
Product Support	http://www.oracle.com/support
Product Forums	http://forums.oracle.com
Product Tutorials	http://www.oracle.com/technology/obe



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This slide lists Oracle websites for Education and Training, documentation, software downloads, articles, support, forums, tutorials, and sample code.

Oracle Database 12c Release 1 (12.1) Documentation



Documentation

This page contains links to the most current documentation for Oracle products. Links are provided for former Sun products. (Use this handy cheat sheet to map former Sun products to Oracle products.)

<ul style="list-style-type: none">DatabaseFusion MiddlewareFusion ApplicationsExalogicExalyticsJavaApplicationsEngineered SystemsServers, Systems Management, Workstations, and Related Hardware	<ul style="list-style-type: none">StorageSystems SoftwareEnterprise ManagementVirtualizationSecure Enterprise TechnologiesOn Demand InfrastructurePreviously Released ProductsLegacy Sun Documentation
--	---

Database

- Database 12c Release 1
- Database 11g Release 2

Oracle Technology Network > Database > Database 12c > Documentation

Overview Downloads Documentation

Oracle Database 12c Documentation

Oracle Database 12c delivers industry leading performance and scalability with a choice of clustered or single-servers running Windows or Linux. Oracle Database 12c features to easily manage the most demanding mission-critical business applications.

Oracle Database, 12c Release 1 (12.1)
E16655-01 zip (915 MB)

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

Oracle University and Error : You are not a Valid Partner user

Oracle Database 12c Release 1 (12.1) Documentation

Documentation Guide Name

Oracle XML DB Developer's Guide 12c Release 1 (12.1)

Oracle Database SQL Language Reference 12c Release 1 (12.1)

Oracle Database PL/SQL Language Reference 12c Release 1 (12.1)

Oracle Database PL/SQL Packages and Types Reference 12c Release 1 (12.1)

Oracle SQL Developer User's Guide Release 3.2

Oracle Database Object-Relational Developer's Guide 12c Release 1 (12.1)

Oracle Database SQLJ Developer's Guide 12c Release 1 (12.1)

Oracle Text Reference 12c Release 1 (12.1)

Oracle Text Application Developer's Guide 12c Release 1 (12.1)



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Navigate to <http://www.oracle.com/pls/db111/homepage> to access the Oracle Database 12c documentation library.

Additional Resources: Using Oracle By Example (OBE) in the Online Learning Library

The OBE series:

- Provides step-by-step instructions on how to perform a variety of new features using Oracle Database 12c
- Reduces the time spent investigating what steps are required to perform a task
- Using practical real world situations, knowledge is gained through valuable hands-on experience and the solutions presented may then be used as the foundation for production implementation, dramatically reducing time to deployment.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Additional Resources: Oracle Online Library (OLL)

You can access the OBEs and other material by using the OLL at: <http://www.oracle.com/oll>

The screenshot shows a web browser window displaying the Oracle Learning Library (OLL) homepage. The URL in the address bar is <https://apex.oracle.com/pls/apex/f?p=44785:1:111055163282286::NO:::>. The page header includes the Oracle logo and navigation links for Home, Products, Search, My Library, and Beta. A banner at the top encourages users to "Like us on Facebook and Follow us on Twitter...". Below the banner, there is a search bar and sections for "Most Popular in Last Month" and "Latest Additions". A tag cloud on the right side shows the tag "jd edwards" with a count of 277. The footer contains the Oracle logo and the copyright notice "Copyright © 2013, Oracle and/or its affiliates. All rights reserved."

The Oracle XML DB Home Page:

<http://www.oracle.com/technetwork/database-features/xmldb/overview/index.html>

The screenshot shows the Oracle Database 11g Release 2 XML DB overview page. At the top, there's a navigation bar with links for Welcome, Account, Sign Out, Help, Select Country/Region, Communities, I am a..., and I want. Below that is a main menu with links for PRODUCTS AND SERVICES, SOLUTIONS, DOWNLOADS, STORE, SUPPORT, TRAINING, PARTNERS, and ABOUT. Under the main menu, the breadcrumb trail shows Oracle Technology Network > Database Features > XML DB > Overview. On the left, there's a sidebar with links for Database EE, XML DB, and XDK. The main content area features the Oracle Database 11g logo and the title "Oracle XML DB". It describes Oracle XML DB as a feature of the Oracle Database, providing high-performance, native XML storage and retrieval technology. It mentions that it fully absorbs the W3C XML data model into the Oracle Database and provides new standard access methods for navigating and querying XML. It highlights advantages over relational database technology and the XML lifecycle management. Below this, there are two bullet points: "Check out the Oracle OpenWorld 2010 presentations on this page" and "XML DB Discussion Forum". A "Technical Information" section follows, listing three white papers: "Best Practices to Get Optimal Performance out of XML Queries (PDF) Jan 2013", "Choosing the Best XMLType Storage Option for Your Use Case (PDF) Jan 2010", and "New Features in Oracle XML DB for Oracle Database 11g Release 2 (PDF) Sept 2009".

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can access the Oracle XML DB home page using the following URL:

<http://www.oracle.com/technetwork/database-features/xmldb/overview/index.html>

This page contains useful information such as white papers, technical presentations, demos, and hands-on practices.

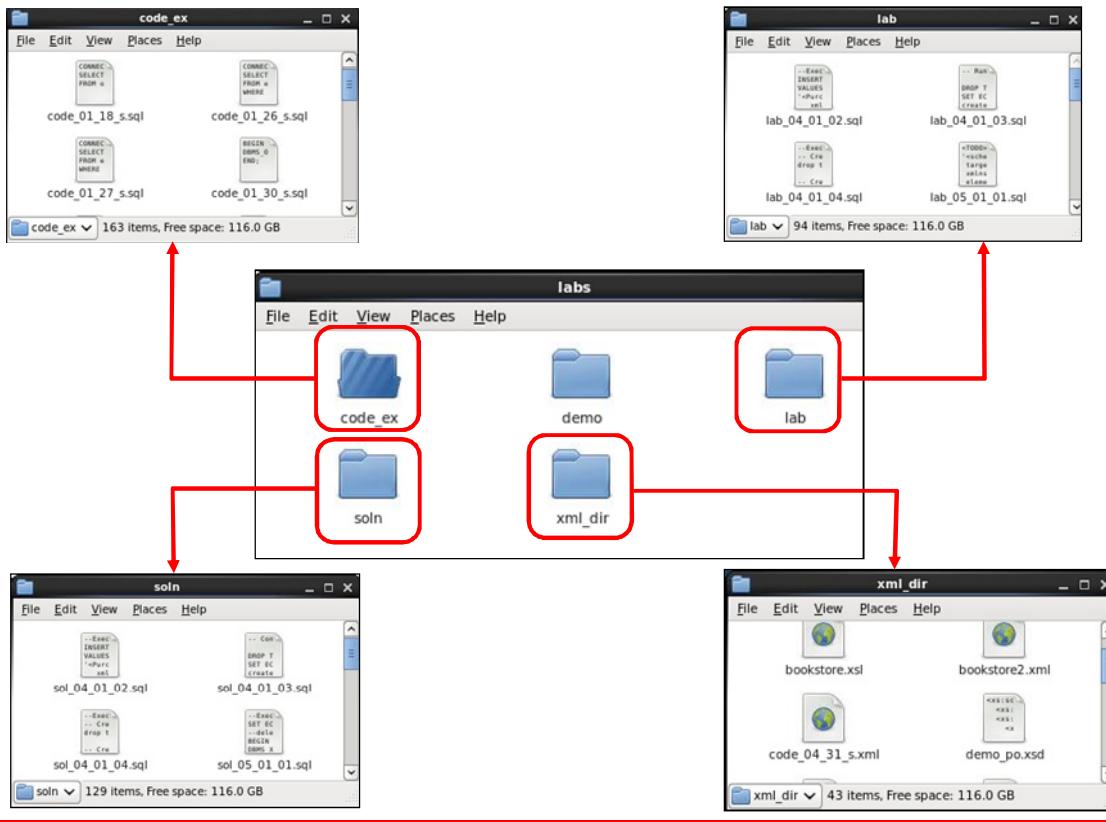
Additional XML Useful Web Sites

Topic	Web Site
W3C XML Schema	http://www.w3.org/XML/Schema
W3C XML Technology	http://www.w3.org/standards/xml/
XML 1.0 W3C Recommendation	http://www.w3.org/TR/xml/
W3C XML Schema	www.w3.org/2001/XMLSchema
XML Schema Part 0: Primer	http://www.w3.org/TR/xmlschema-0/
W3C XML Schema Structures	http://www.w3.org/TR/xmlschema-1/
W3C XML Schema Datatypes	http://www.w3.org/TR/xmlschema-2/
W3C XML XPath Version 1.0	http://www.w3.org/TR/xpath/
XPath Tutorial	http://www.zvon.org/xxl/XPathTutorial/General/examples.html
XQuery Tutorial	http://www.w3.org/TR/xquery/
W3C Namespaces in XML	http://www.w3.org/TR/xml-names/
Document Object Model (DOM)	http://www.w3.org/DOM/
XSL Transformations (XSLT)	http://www.w3.org/TR/xslt



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Course Scripts in the /home/oracle/labs Folder



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The `/home/oracle/labs` folder contains five folders: `solns`, `lab`, `code_ex`, `xml_dir`, and `demo`.

The practices and solutions to the lessons are found in the `/home/oracle/labs/solns` and `/home/oracle/labs/lab` folders. The solutions are named after the lesson number and practice question. For example, the `sol_18_02_01.sql` script is the solution code for question 1 in Practice 18-2 in the case study. Similarly, the `lab_18_03_05.sql` script is the practice used in question 5 in Practice 18-3.

The `code_ex` folder contains the scripts for most (but not all) of the code examples in the lessons. The code examples are named after the lesson number and page number.

The `demo` folder contains a few demo scripts. The demo scripts are named based on their functionality. For example, the `demo_regschema.sql` demonstrates how to register an XML schema.

The `xml_dir` folder contains some structures and files that are used in this course. In order for you to insert XML documents in `XMLType` tables in this course, a SQL directory object that points to the `xml_dir` directory that contains the files to be processed was already created as follows:

```
CREATE DIRECTORY xml_dir AS '/home/oracle/labs/xml_dir';
```

Summary

In this lesson, you should have learned how to:

- Discuss the course objectives, prerequisites and suggested prerequisites, students' introductions, and agenda
- List the database and XML schemas used in this course
- List the appendices used in this course
- Identify the relevant documentation and other resources



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 1-1: Overview

This practice covers the following topics:

- Starting SQL Developer, creating and testing a new database connection, and connecting to the database
- Using the SQL Worksheet



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The goal of this practice is to become familiar with the SQL Developer tool to execute SQL statements.

Note

- All written practices use SQL Developer as the development environment. Although it is recommended that you use SQL Developer, you can also use the SQL*Plus or JDeveloper environments that are available in this course.
- All practices and practice solutions in this course assume that you run the scripts by using the SQL Worksheet area in SQL Developer.

Practice 1-2: Overview

This practice covers the following topics:

- Using your web browser to access and review some of the useful XML resources and documentation
- Bookmarking some of the resources for easier access



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The goal of this practice is to access and review some of the useful XML resources and documentation.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

2

XML Basic Review

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe Extensible Markup Language (XML) and its structure and components
- Review Document Type Definition (DTD)
- Describe an XML Namespace
- Review and use an XML Schema
- Define and use the XML Path language (XPath)
- Define and use XQuery
- Review XML Stylesheet Language (XSL) transformations



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

What Is XML

- XML stands for EXtensible Markup Language.
- XML is a markup language just like HTML.
- XML was designed to carry data, not to display data.
- XML tags are not predefined like HTML. You must define your own custom tags.
- XML is designed to be self-descriptive.
- XML is a W3C Recommendation.
- XML is the most common tool for data transmissions between all sorts of applications.
- XML data is stored in plain text format. This makes it a software and hardware independent tool for storing and carrying information.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The Difference Between XML and HTML

- XML is not a replacement for HTML.
- XML and HTML were designed with different goals:
 - XML was designed to transport and store data, with focus on what data is.
 - HTML was designed to display data, with focus on how data looks.
- HTML is about displaying information, while XML is about carrying information.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XML Documents Form a Tree Structure

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

```
<employees>
  <employee>
    <employee_id>120</employee_id>
    <last_name>Weiss</last_name>
    <salary>8000</salary>
  </employee>
  <employee>
    <employee_id>121</employee_id>
    <last_name>Fripp</last_name>
    <salary>8200</salary>
  </employee>
</employees>
```

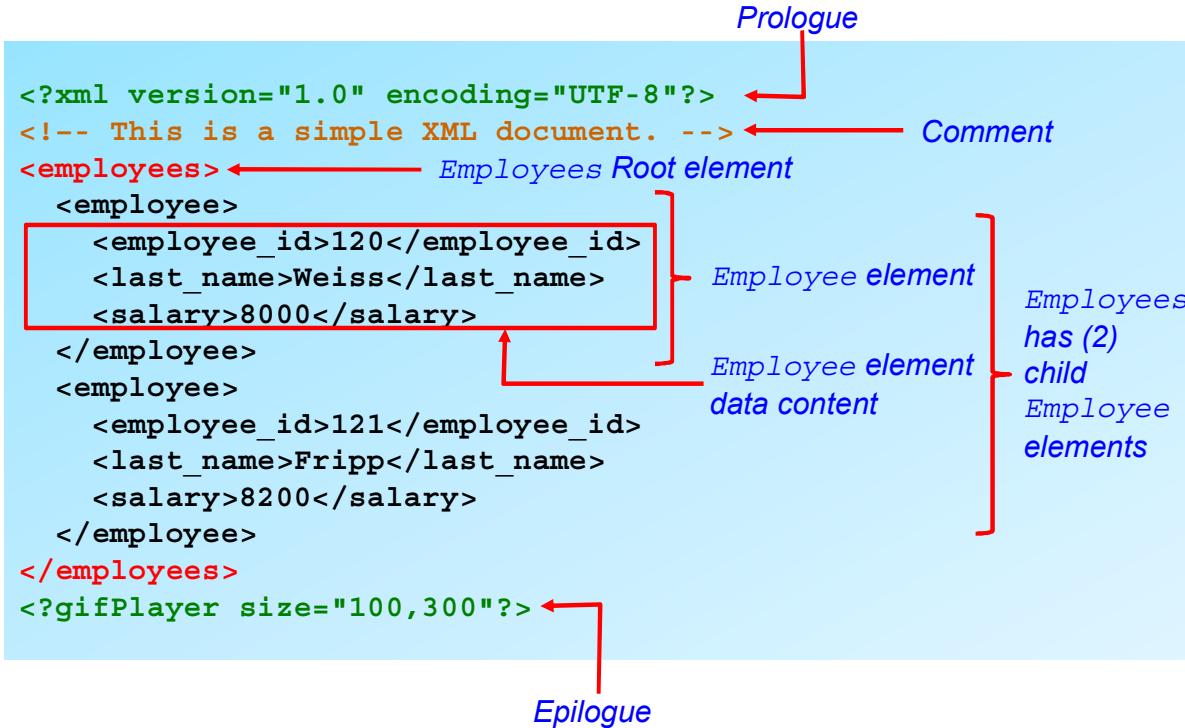


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XML documents must contain a root element. This element is “the parent” of all other elements. The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree. All elements can have subelements (child elements).

The slide example displays an employee information, which is self-descriptive. It contains an employee information wrapped in custom tags. XML has no predefined tags like HTML.

An Example of a Simple XML Document



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example of a simple XML document uses nested elements to describe employee data. Elements are identified by tag names such as `employee`, `employee_id`, and `last_name`. Tag names are distinguishable as markup, rather than data, because they are surrounded by angle brackets (`<` and `>`). In XML, an element includes a start tag (`<employees>`), an end tag (`</employees>`), and all the markup and character data contained between those tags. Tag names are case-sensitive (must be identical).

An XML document contains the following parts:

- The prologue, which may contain the following information: XML declaration, document type definition (DTD), which is required only to validate the document structure, and processing instructions and comments, which are optional
- The root element, which is also called the “document element” and contains all other elements
- Any child elements (two employees in the example)
- An epilogue, which contains processing instructions and comments. Processing instructions give commands or information to an application that processes the XML data.

XML Elements and Attributes

An XML document contains XML Elements. An element can contain other elements, text, attributes, or a combination of the preceding.

```
<bookshop>
  <book category="CHILDREN">
    <title>Love You Forever</title>
    <author>Robert Munsch</author>
    <year>1995</year>
    <price>4.98</price>
  </book>
  <book category="COMPUTERS">
    <title>XML: Visual QuickStart Guide</title>
    <author>Kevin Howard Goldberg</author>
    <year>2008</year>
    <price>23.17</price>
  </book>
</bookshop>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

An XML document contains XML Elements. An XML element is everything from (including) the element's start tag to (including) the element's end tag.

In the example above, `<bookshop>` and `<book>` have element contents, because they contain other elements. `<book>` also has an attribute (`category="CHILDREN"`). `<title>`, `<author>`, `<year>`, and `<price>` have text content because they contain text.

XML Markup Syntax Rules for Elements

- There is one root element, which is sometimes called the top-level or document element.
- All elements:
 - Must have matching start and end tags, or be a self-closing tag (an empty element)
 - Can contain nested elements so that their tags do not overlap
 - Have case-sensitive tag names that are subject to naming conventions (starting with a letter, no spaces, and not starting with the letters `xml`)
 - May contain white space (spaces, tabs, new lines, and combinations of them) that is considered part of the element data content



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Every XML document must contain one root element (top-level or document element). XML documents are hierarchical in structure with elements nested within others forming a document tree. The start and end tags for elements must not overlap. For example:

```
<employee>
  <first_name>Steven<last_name>king</first_name></last_name>
</employee>
```

Here, the `<last_name>` element overlaps the `<first_name>` element. This is not permissible. The correct form is:

```
<employee>
  <first_name>Steven</first_name><last_name>king</last_name>
</employee>
```

Element start and end tag names must be identical; that is, they are case-sensitive. For example, `<Employee>`, `<employee>`, and `<EMPLOYEE>` are distinct and different tag names.

XML Syntax Rules

- XML documents must have a root element.
- All XML elements must have a beginning and closing tag.
- XML tags are case-sensitive.
- XML elements must be properly nested.
- XML attribute values must be quoted.
- XML documents can contain comments.
- White space is preserved in XML.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Element tag names must start with a letter or an underscore (_) but not with numeric or punctuation characters. Numeric, dash (-), and period (.) characters are allowed after the first letter, but not white space. Tag names cannot start with the xml letter sequence or any case-sensitive combination thereof, such as XML and Xml. White space, including new lines, are considered part of the data; that is, no stripping of white space is done. However, XML Parsers treat end-of-line characters as a single line-feed.

XML Attributes

An XML attribute is a name-value pair that:

- Is specified in the start tag, after the tag name

```
<?xml version="1.0" encoding="UTF-8"?>
<employees>
  <employee id="100" name='Rachael O'Leary'>
    <salary>1000</salary>
  </employee>
</employees>
```

- Has a case-sensitive name
- Has a case-sensitive value that must be enclosed within matching single or double quotation marks
- Provides additional information about the XML document or XML elements



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Attributes are simple name-value pairs that are associated with a particular element. XML attributes must be specified after the start tag of an element or after the tag name of an empty element.

Example: `<employee id="100" email="SKING" />`

Attribute names are case-sensitive and follow the naming rules that apply to element names. In general, spaces are not used, but are allowed, on either side of the equal sign. Attribute names should be unique within the start tag.

The attribute values must be within matching quotation marks, either single or double. The example in the slide shows the `employee id` attribute value enclosed within double quotation marks and the `name` attribute value within single quotation marks. In the latter case, the `'` entity must be used to include the apostrophe (single quotation mark) character in the `name` value.

Attributes provide additional information about the XML document's content or other XML elements.

Well-Formed XML Documents

Every XML document must be well-formed:

- An XML document must have one root element.
- An element must have matching start and end tag names unless it is an empty element.
- Elements can be nested but cannot overlap.
- All attribute values must be quoted.
- Attribute names must be unique in the start tag of an element.
- Comments and processing instructions do not appear inside tags.
- The < or & special characters cannot appear in the character data of an element or attribute value.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

An XML document must be well-formed to guarantee that it is correctly structured and adheres to the rules defined in the slide. The slide contains a list of the most common rules for well-formed documents, but it is not a complete list.

Note

The less-than (<) and ampersand (&) characters are special in XML and cannot appear as themselves within the character data part of an element, or the value of an attribute. In character data and attribute values, an XML Parser recognizes:

- The less-than (<) sign as a character that introduces the start tag of another element
- The ampersand (&) as an escape character before an entity name terminated by a semicolon (;)

Therefore, to include special characters in the character data of an element or attribute value, you must use built-in XML entity names for the special characters. For example, use < to include the less-than character, and & for the ampersand character. The XML Parser replaces the entity reference with its textual or binary representation, as discussed earlier in this lesson.

The XML 1.1 specification requires the XML declaration to appear at the beginning of the document. However, the declaration is optional in XML 1.0.

Document Type Definition (DTD)

A DTD:

- Is the grammar for an XML document
- Contains the definitions of:
 - Elements
 - Attributes
 - Entities
 - Notations
- Contains specific instructions that the XML Parser interprets to check document validity
- May be stored in a separate file (external)
- May be included in the XML document (internal)



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A DTD provides a list of elements contained in an XML document that collectively specifies the structure of the document. A DTD defines the set of markup items used in an XML document:

- Elements
- Attributes and their permissible values
- Entities (user-defined)
- Notations

Note: Notations are not frequently used.

The XML engine interprets the markup defined in a DTD to check if the document is valid. For example, a DTD may specify that a department must have exactly one department identification number and one name, but may have multiple employee elements. An XML document indicates to an XML Parser whether it is associated with a DTD and where to locate the DTD.

A DTD may be found internally (inline) in the document, or externally, identified by a uniform resource locator (URL), which can be a file on disk. If the XML Parser does not encounter errors, the XML document is guaranteed to be consistent with the definition.

Why Validate an XML Document?

- Well-formed documents satisfy XML syntax rules, and not the business requirements about the content and structure.
- Business rules often require validation of the content and structure of a document.
- XML documents must satisfy the structural requirements imposed by the business model.
- A valid XML document can be reliably processed by XML applications.
- Validation can be done by using a DTD or an XML schema.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A well-formed XML document is one that meets all the rules of the XML specification. Your business may require the validation of the actual structure and content of a document. In addition to conforming to the XML rules, your document should also satisfy the structural requirements imposed by your business model. Here is an example:

Each `<employee>` element must consist of `<employee_id>`, `<first_name>`, `<last_name>`, `<email>`, `<phone_number>`, and `<department_id>` elements. If an `<employee>` element misses any of these elements, it is not valid. You may also need to verify whether these elements have a valid value such as a valid `employee_id` and `department_id`.

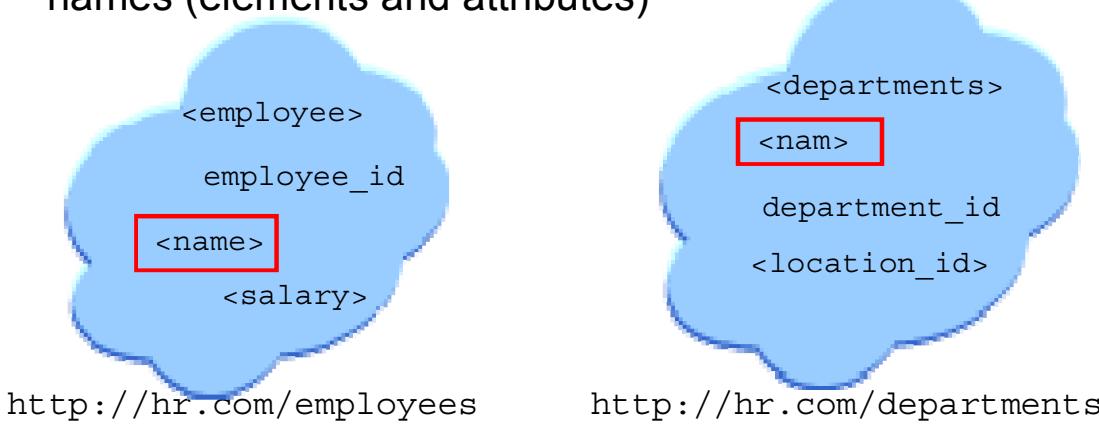
You can perform all these validations by using a DTD. The DTD was the first type of mechanism available for validating XML document structure and content. However, it lacked the capability to perform data type validations on the content. The XML Schema recommendation supports validation for different data types, and is rapidly replacing DTDs as a way to validate the XML document structure and content.

Note: Theoretically, anything that has not been explicitly permitted in a DTD is forbidden. However, some parsers do not always enforce the rules defined by a DTD.

XML Namespace

XML Namespaces provide a method to avoid element name conflicts. An XML Namespace:

- Is identified by a case-sensitive internationalized resource identifier (IRI) reference (URL or URN)
- Provides universally unique names for a collection of names (elements and attributes)



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

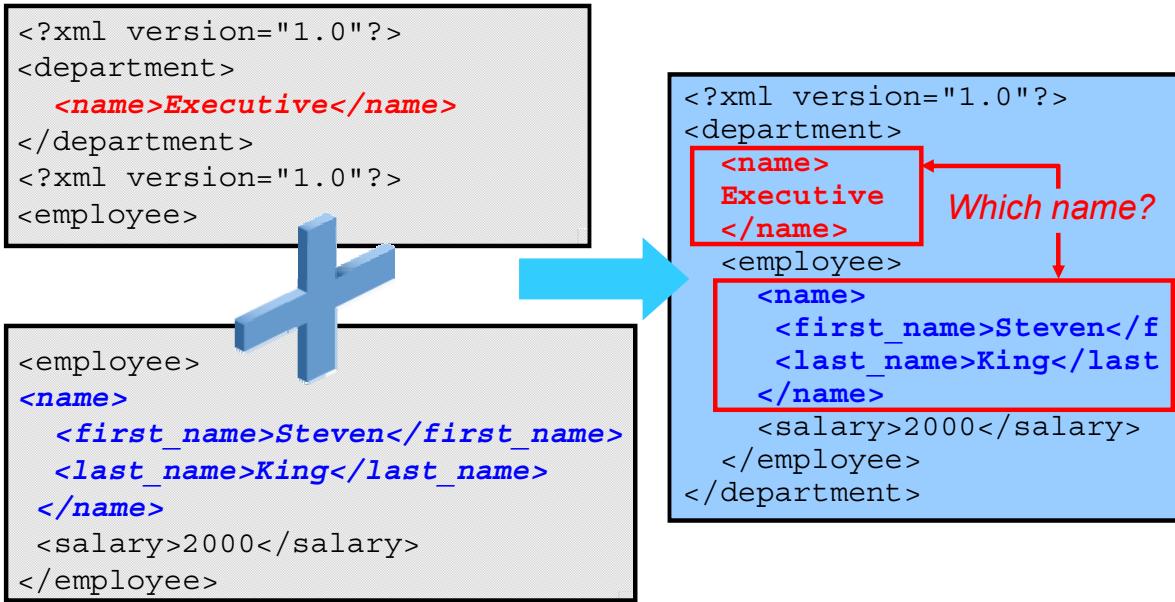
An XML Namespace:

- Is identified by an internationalized resource identifier (IRI), which is a case-sensitive string of characters identifying a resource
- Provides a universally unique name for a collection of XML names made of element and attribute names

The example in the slide shows two collections of XML element and attribute names, each identified by a unique XML Namespace url. If the XML Namespace is not used, the `<name>` element for an employee would be indistinguishable from the `<name>` element of the department. Applying the XML Namespace qualifies each `<name>` element by making them unambiguous, especially if the documents are merged.

Why Use XML Namespaces?

Using an XML Namespace resolves name conflicts in an XML document.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XML Namespaces are designed to provide universally unique names for elements and attributes. XML Namespaces:

- Resolve name ambiguity and collision problems that can occur when XML document fragments are combined, or multiple elements have similar type and attribute names in the same XML document
- Allow code modules to be invoked for specific elements and attributes

If two companies are interchanging XML messages, they must come to a common agreement on the meaning of the element and attribute names in the messages. This can be achieved by two means:

- Defining the meaning, format, and domain of every element and attribute needed
- Recognizing the element and attribute names without ambiguity

The first can be accomplished by using an XML Schema definition. The second is solved by using XML Namespaces to qualify element names.

The examples in the slide illustrate the combining of a `<department>` element with one of its `<employee>` elements. Each document is different and defines a `<name>` element. Each `<name>` element has a different content model and must be interpreted by an application in a different way. The problem does not arise when the elements exist in separate documents.

Declaring XML Namespaces: The XMLNS Attribute

- You can avoid name conflicts in XML by using a name prefix.
- When you use prefixes in XML, you must declare a namespace for the prefix.
- The namespace is defined by the `xmlns` attribute in the start tag of an element.
- The namespace declaration has the following syntax:
`xmlns:prefix="URI"`.
- When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XML Namespace Declarations and Default Namespaces: Example

```

<?xml version="1.0"?>
<department xmlns="urn:hr:department-ns" → Default namespace
             xmlns:emp="urn:hr:employee-ns"> ] -->
  1   <name>Executive</name>

  <emp:employee> 2
    <emp:name> ] 
      <emp:first_name>Steven</emp:first_name>
      <emp:last_name>King</emp:last_name>
    </emp:name>
  </emp:employee>
  <emp:employee>
    <emp:name>
      <emp:first_name>Neena</emp:first_name>
      <emp:last_name>Kochhar</emp:last_name>
    </emp:name>
  </emp:employee>
</department>

```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide shows an example of a `<department>` element containing two `<employee>` elements. The `<department>` element shows an example of declaring two XML Namespaces: a default and one with the `emp` prefix. The `<name>` element is used for:

- The department name containing the `Executive` text
- The employee name containing the `<first_name>` and `<last_name>` child elements

Without using XML Namespaces in the document, the `<name>` element will be ambiguous to a processor and can possibly be treated as the same type, even though the department and employee names are semantically and structurally different.

Using XML Namespaces removes the ambiguity for each `<name>` element, which allows them to be processed differently by an XML application. In the example in the slide:

- The unqualified `<department>` and `<name>` elements are implicitly qualified by the default namespace, `urn:hr:department-ns`, declared in the start tag of the `<department>` element
- All `<employee>` elements and their children, including the `<emp:name>` element, are explicitly qualified with the `emp` prefix, which is associated with the `urn:hr:employee-ns` XML Namespace

XML Schema

An XML schema:

- Describes the structure of an XML document
- Is an XML-based alternative to DTD
- Is an XML language that defines and validates the structure of XML documents
- Is stored in an XML schema Document (XSD)
- Defines components such as:
 - Simple and complex type definitions
 - Element and attribute declarations
- Builds on the DTD functionality while providing XML Namespaces, built-in, simple, and complex data types support



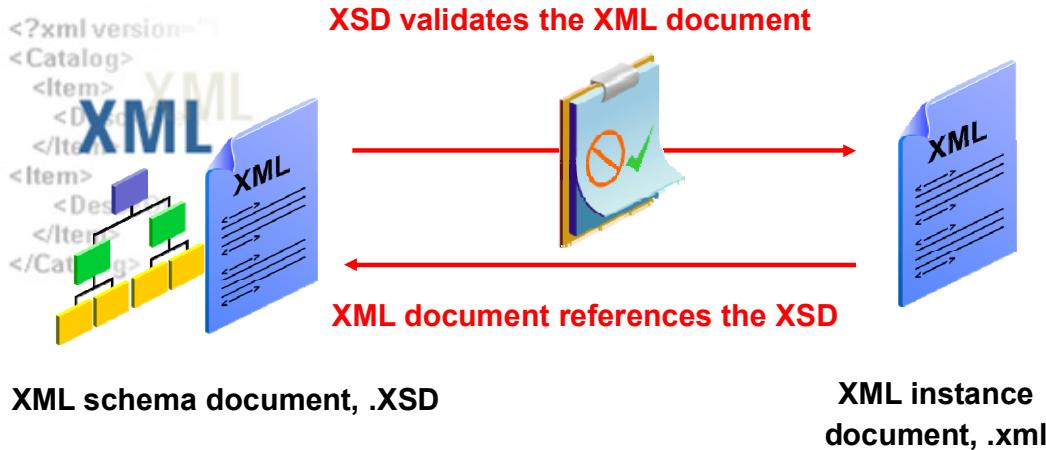
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The W3C XML Schema Definition Language is an XML language (or vocabulary) that is used in an **XML schema document** (XSD) for describing and constraining the content of XML documents. The XSD is used to validate the structure of an XML document.

An XML document, whose structure is based on the definitions in an XML schema, is called an **instance document** of that XML schema. The introduction of XML Schema allowed XML technology to represent data types in a standard format. The data types give a precise way of specifying the type of content that can be held in the elements and attributes of an XML document. A document type definition (DTD) provided no mechanism for specifying data types in a way that a database user may require. The XML Schema definition file builds on the DTD functionality while providing XML Namespace and data type support.

XML Schema



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

An XML schema document is stored separately from the XML instance document that it describes and validates.

XML Schema Document: Example

- A simple XML schema uses:
 - A required XML Namespace string, with an `xs` prefix, `http://www.w3.org/2001/XMLSchema`
 - The `<schema>` element as its document root
 - The `<element>` element to declare an element

```
<?xml version="1.0"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="departments" type="xs:string"/>
</xs:schema>
```

XSD

- A valid XML instance document:

```
<?xml version="1.0"?>
<! -- The element cannot contain child elements -->
<departments>Finance</departments>
```

XML

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows a simple XML schema document that declares a single element called `departments`. The XML Schema Document (XSD) uses the required XML Namespace value `http://www.w3.org/2001/XMLSchema`, which is assigned the `xs` namespace prefix. The `xs` prefix is used to qualify the `schema`, `element`, and `string` names to ensure that the XSD document structure conforms to the W3C XML Schema language recommendation; that is, the XSD itself is a valid document.

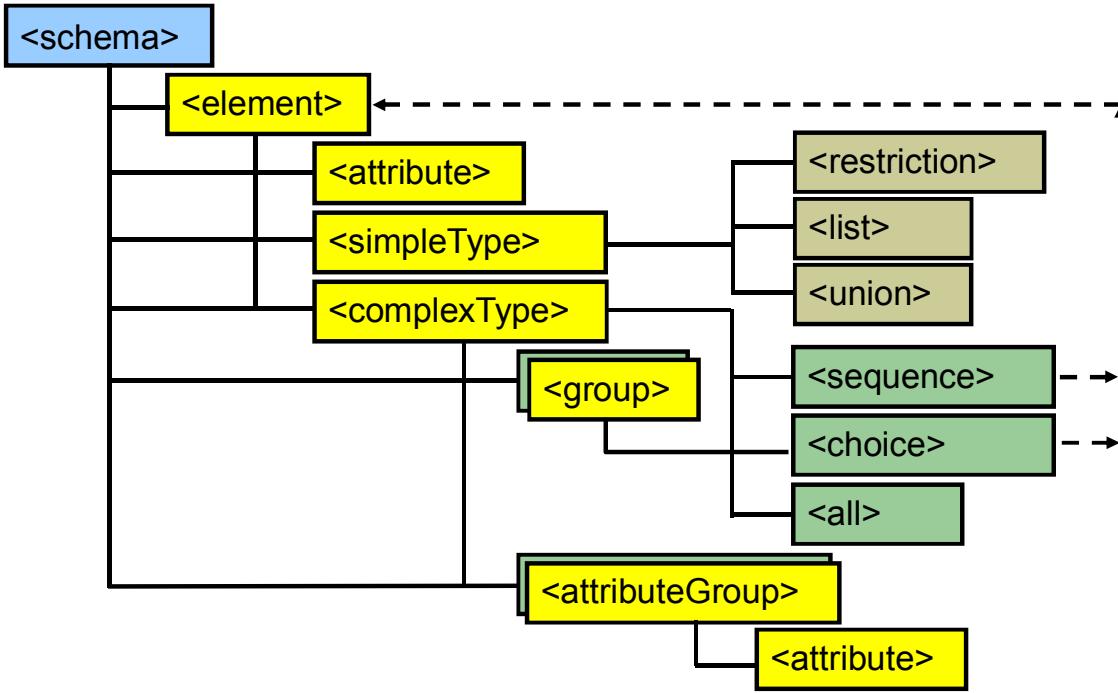
The XML Schema language elements used in the example are:

- The `<xs:schema>` element, which is the root element for the XSD, and contains the definitions for the structure of the XML instance document
- The `<xs:element>` element, which declares a root element name, `departments`, for the XML instance document, an example of which is shown below the XSD code
- The `<xs:string>` value, which is set as the data type for the contents of the `departments` element in the XML instance document. In this case, the data can be any string but not markup; that is, no child elements are permitted.

Although the example is a very simple XML Schema document, the resulting XML instance document structure that can be created is not very useful.

Note: Using a namespace prefix, such as `xs` or `xsd`, is recommended but not required.

Components of an XML Schema



ORACLE

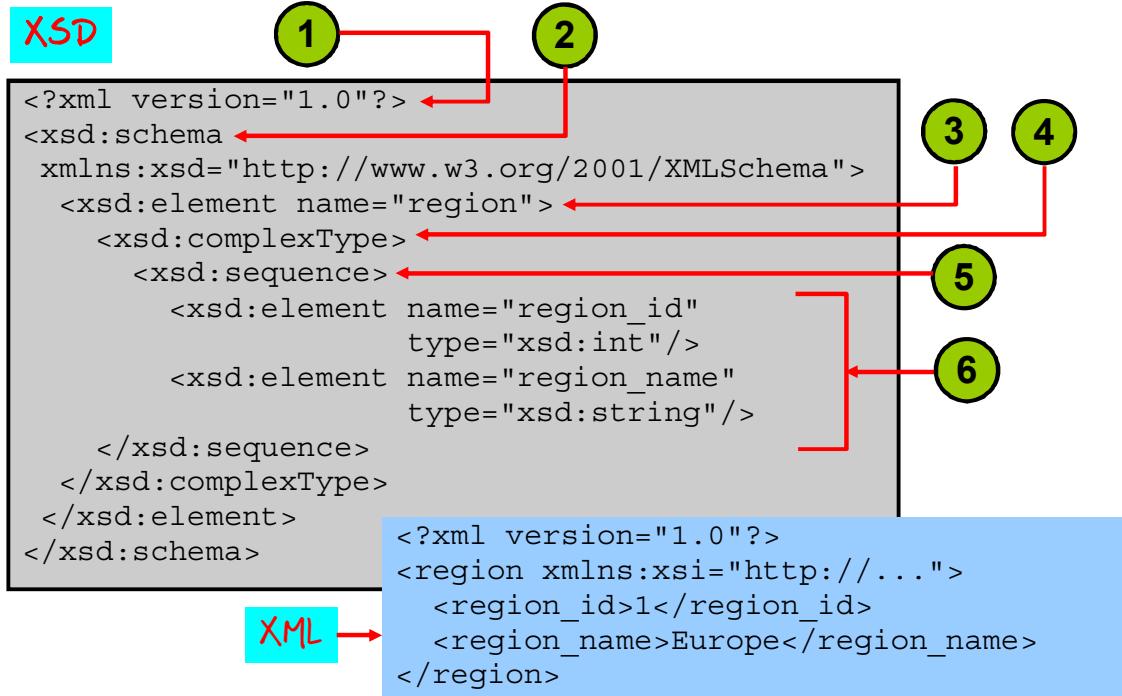
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The root of an XML schema document is the `<schema>` component, which can contain one or more declarations for:

- An `<element>` for XML elements, which can contain `<attribute>`s or be structured as `<simpleType>` or `<complexType>`
- An `<attribute>` representing a named attribute XML element
- A `<simpleType>`, which is an atomic built-in data type, or structured as a `<list>` or `<union>` of other components such as `<enumeration>` components
- A `<complexType>`, which defines a complex structure that may be composed of other components such as a `<sequence>`, `<choice>`, or `<group>`. For example, a `<complexType>` is needed to declare a structure with child elements, and can define type structures used for deriving other types or used as a user-defined type.
- A `<group>`, which names a structure composed of a `<sequence>`, `<choice>`, or an `<all>` component
- An `<attributeGroup>` defining a list of attributes

Note: An XML Schema contains more component types than is discussed in this course, which covers the `<element>`, `<attribute>`, `<simpleType>`, `<complexType>`, and `<sequence>` components. The `<attribute>` item is duplicated for clarity in the slide.

XML Schema Components: Example



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example XML schema contains the following components:

1. The XML declaration
2. The `<schema>` root element with the namespace declaration `xmlns:xsd="http://www.w3.org/2001/XMLSchema"` to indicate that the elements used in the XML schema document, which are specified in the W3C XML Schema Language recommendation, define the structure of an XML instance document. The namespace prefix, such as `xsd`, is optional but recommended. When the prefix is declared, the XML schema elements must use the prefix: for example, `<xsd:schema>`.
3. The `<xsd:element>` declaration within the `<schema>` element, which specifies a root element name for the XML instance document: for example, `region`
4. The `<xsd:complexType>` definition within `<xsd:element>`, which is needed to indicate that the `<region>` element contains a sequence of child elements named `<region_id>` and `<region_name>`
5. `<xsd:sequence>`, which defines the order of occurrence of the two element declarations that it contains, and forms the content model for the `region` element
6. The two `<xsd:element>` declarations for the child elements `region_id` and `region_name`, based on the built-in `int` and `string` data types, respectively

XML Path Language

XPath is a language for finding information (elements and attributes) in an XML document. XPath:

- Treats XML documents as trees of nodes.
- Uses path expressions to select nodes or node sets in an XML document
- Is named after the path notation it uses for navigating through the hierarchical structure of an XML document
- Uses a compact, non-XML syntax to form expressions for use in URI and XML attribute values
- Fully supports XML Namespaces
- Contains a library of standard functions



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XML Path Language (XPath) is primarily used for addressing parts (nodes) of an XML document. XPath models an XML document as a tree of nodes, and expresses a navigation path through the hierarchical structure of an XML document. XPath:

- Is named after the path notation it uses for navigating through the structure of an XML document
- Uses a compact, non-XML syntax to form expressions that are used within URI and XML attribute values
- Facilitates the manipulation of string, number, and Boolean values
- Operates on the abstract, logical structure of an XML document called the *document data model*, rather than its surface syntax

XPath Terminology

- Nodes
 - Elements, attribute, text, namespace
 - Processing-instructions, comments, document nodes
- Family Relationship
 - Parent, children, siblings, ancestors, and descendants

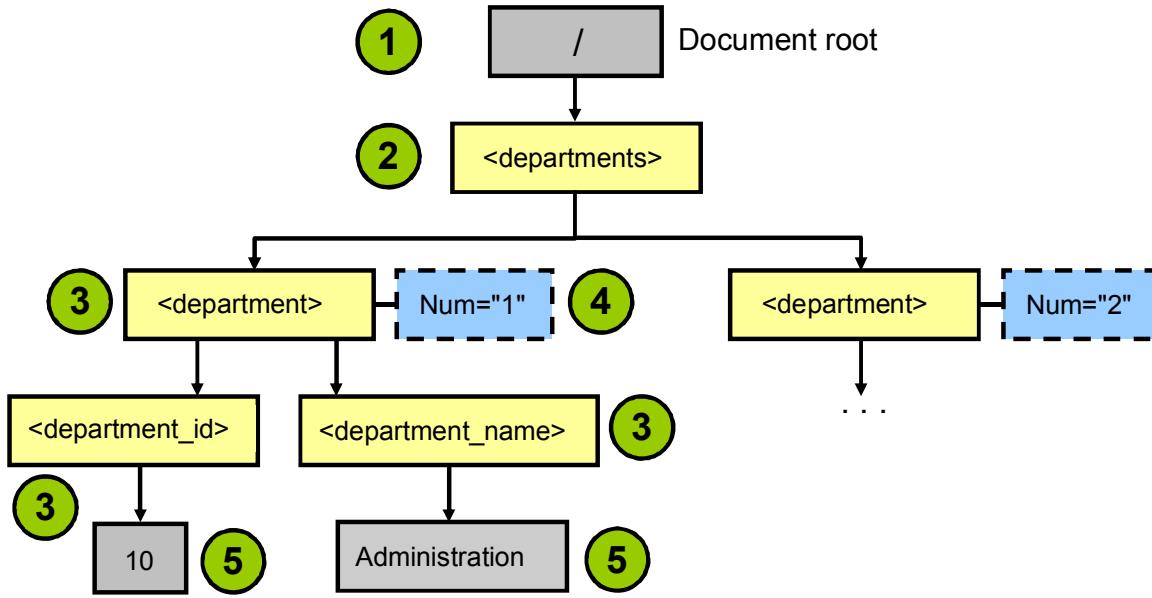
```
<bookshop>
  <book category="CHILDREN">
    <title>Love You Forever</title>
    <author>Robert Munsch</author>
    <year>1995</year>
    <price>4.98</price>
  </book>
</bookshop>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XML documents are treated as trees of nodes. The topmost element of the tree is called the root element. In XPath, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document nodes. In the slide example, some of the nodes are `<bookshop>` (root element node), `<book>`, `<title>`, `<author>`, `<year>`, and `<price>`. `Category="CHILDREN"` is an attribute node. "CHILDREN", Love you Forever, 1995, and 4.98 are atomic values.

XPath Model As a Tree (Partial Example)



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Again, XPath models an XML document as a tree of nodes. The partial graphic in the slide depicts the different types of nodes as seen by XPath in an XML document:

1. Document root, which is a virtual document root that is the parent of the entire XML document containing the XML declaration, the root element and its children, comments, and processing instructions at the beginning and end of the document
2. Root element (for example, the `<departments>` element)
3. Element nodes (for example, the `<department>`, `<department_id>`, and `<department_name>` elements. The root element is also an element node.)
4. Attribute nodes (for example, the `num= "1"` node)
5. Text nodes (for example, the nodes containing the text `10` and `Administration`)

XPath can also address comment nodes, processing instruction nodes, and namespace nodes, which have not been shown in the diagram. XPath defines a way to compute a string value for each type of node, some of which have names. Because XPath fully supports the XML Namespaces Recommendation, the node name is modeled as a pair that is known as the **expanded name**, which consists of a **local part**, and a **namespace URI**, which may be null.

Note: In the XPath model, the document root is not the same as the root element node.

XPath Model as an XML Document

```
<?xml version='1.0' encoding="UTF-8"?>
<departments>
  <department num="1">
    <department_id>10</department_id>
    <department_name>Administration</department_name>
    <manager_id>200</manager_id>
    <location_id>1700</location_id>
  </department>
  <department num="2">
    <department_id>20</department_id>
    <department_name>Marketing</department_name>
    <manager_id>201</manager_id>
    <location_id>1800</location_id>
  </department>
  <department num="3">
    <department_id>30</department_id>
    <department_name>Purchasing</department_name>
    <manager_id>114</manager_id>
    <location_id>1700</location_id>
  </department>
  <department num="4">
    <department_id>40</department_id>
    <department_name>Human Resources</department_name>
    <manager_id>203</manager_id>
    <location_id>2400</location_id>
  </department>
</departments>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XPath Expressions

An XPath expression:

- Is the primary construct in XPath
- Is evaluated to yield an object whose type can be:
 - A node-set, a Boolean, a number, or a string
- Is evaluated within a context (starting point) that includes:
 - A node called the context node
 - The context position and the context size
 - A function library, among others
- Can be a location path. This is the most important and commonly used expression type.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XPath language provides several kinds of expressions that may be constructed from keywords, symbols, and operands. Generally, an operand forms another kind of expression. An XPath expression is evaluated to yield an object of the following basic types:

- **A node-set:** An unordered collection of nodes, excluding duplicates
- **A Boolean:** A true or false result
- **A number:** A floating-point number
- **A string:** A sequence of Universal Coded Character Set (UCS) characters

XPath expressions are evaluated in a context, which consists of:

- A node, called the **context node**
- A pair of nonzero positive integers known as the context position and context size
- A set of variable bindings
- A function library
- A set of namespace declarations in scope for the expression

XPath is a strongly typed language such that the operands of various expressions, operators, and functions must conform to designated types. XPath is a case-sensitive expression language.

Location Path Expression

- All legal XPath codes can be called expressions.
- An XPath expression that returns a node-set is called a *location path*.
- The location path expression is one of the following:
 - A relative location path:
 - Is made up of one or more location steps that specify navigation directions to the nodes in an XML document
 - Is relative to the starting point (called the context node)
 - An absolute location path:
 - Always starts from a standard point, the root node, /, followed by a relative location path



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The location path is one of the most important XPath expressions that gives an application navigation directions in an XML document to locate a set of nodes. A **location path** can be either of the following:

- An **absolute location path**, which starts with a slash (/) and is followed by a relative location path
- A **relative location path**, which is made up of a sequence of one or more **location steps** that are separated by a slash (/). Location steps are composed from left to right selecting or navigating to a set of nodes, relative to the **context node**.

Selecting Nodes

Expression	Description
nodename	Selects all nodes with the name "nodename"
/	Selects nodes from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are in the document
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes
*	Matches any element node
@*	Matches any attribute node



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XPath uses path expressions to select nodes in an XML document. The node is selected by following a path. The table in the slide lists some of the commonly used path expressions.

Class Review: Examine the XML Document

```
<bookshop>
  <book category="CHILDREN">
    <title>Love You Forever</title>
    <author>Robert Munsch</author>
    <year>1995</year>
    <price>4.98</price>
  </book>

  <book category="COMPUTERS">
    <title>XML: Visual QuickStart Guide</title>
    <author>Kevin Howard Goldberg</author>
    <year>2008</year>
    <price>23.17</price>
  </book>
</bookshop>
```

Expression	Result
bookshop	Selects all nodes with the name bookshop.
/bookshop	Selects the root element bookshop.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Class Review: Examine the XML Document

```
<bookshop>
  <book category="CHILDREN">
    <title>Love You Forever</title>
    <author>Robert Munsch</author>
    <year>1995</year>
    <price>4.98</price>
  </book>

  <book category="COMPUTERS">
    <title>XML: Visual QuickStart Guide</title>
    <author>Kevin Howard Goldberg</author>
    <year>2008</year>
    <price>23.17</price>
  </book>
</bookshop>
```

Expression	Result
//book	Selects all book elements no matter where they are in the document.
//@category	Selects all attributes that are named category.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Class Review: Examine the XML Document

```
<bookshop>
  <book category="CHILDREN">
    <title>Love You Forever</title>
    <author>Robert Munsch</author>
    <year>1995</year>
    <price>4.98</price>
  </book>

  <book category="COMPUTERS">
    <title>XML: Visual QuickStart Guide</title>
    <author>Kevin Howard Goldberg</author>
    <year>2008</year>
    <price>23.17</price>
  </book>
</bookshop>
```

Expression	Result
/bookshop/*	Selects all the child nodes of the bookshop element.
//*	Selects all elements in the document.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Class Review: Examine the XML Document

```
<bookshop>
  <book category="CHILDREN">
    <title>Love You Forever</title>
    <author>Robert Munsch</author>
    <year>1995</year>
    <price>4.98</price>
  </book>

  <book category="COMPUTERS">
    <title>XML: Visual QuickStart Guide</title>
    <author>Kevin Howard Goldberg</author>
    <year>2008</year>
    <price>23.17</price>
  </book>
</bookshop>
```

Expression	Result
//title [@*]	Selects all title elements which have any attribute.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Location Path Expression: Example

XPath expression

/departments/department/department_id

```
<?xml version="1.0"?>
<departments>
  <department num="1">
    <department_id>10</department_id>
    <department_name>Administration</department_name>
  </department>
</departments>
...
```

XML

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The location path expression shown in the slide is an absolute path that uses the document root as the context node. An absolute location path always starts with a slash (/), and is followed by a relative location path that comprises the following location steps:

1. Go to the `<departments>` root element relative to document root.
2. Go to the first child `<department>` element of the `<departments>` node.
3. Go to the `<department_id>` child element of the `<department>` node.

If the sample XML document includes more than one `<department>` element and child `<department_id>` element, the XPath expression matches multiple `<department_id>` nodes or a set of `<department_id>` nodes. The result is the entire element from the start to the end tags, plus the data between them. As shown in the highlighted portion of the slide, the result includes `<department_id>10</department_id>`.

XPath Node Test Types

An XPath node test can be:

- A node-name such as:
 - The element name
 - The attribute name if prefixed with an @ symbol
 - A qualified name with a namespace prefix
- An asterisk (*)
- The text() type
- The processing-instruction() type
- The comment() type
- The node() type
- Separated by the vertical bar (|) to form multiple matches



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The node type that is specified as the node test for a location step can be:

- A node-name representing either a qualified or an unqualified element name. If it is prefixed with an @ symbol, the node-name represents an attribute name.
- An asterisk (*) to match all element or attribute node names
- The text() type to select text nodes
- The processing-instruction() type for processing instruction nodes
- The comment() type for obtaining the comment nodes
- The node() type to match any of the preceding nodes

When the vertical bar (|) is used as a separator, node tests can be combined to create a rule that matches multiple node types. For example, using /text() | comment() matches either the text or comment nodes of the root element.

Note: If the nodes in the XML document declare an XML Namespace with a prefix, the node name must be qualified by the namespace prefix in the XPath expression.

Abbreviated XPath Expression Examples: The XML Document

employees.xml

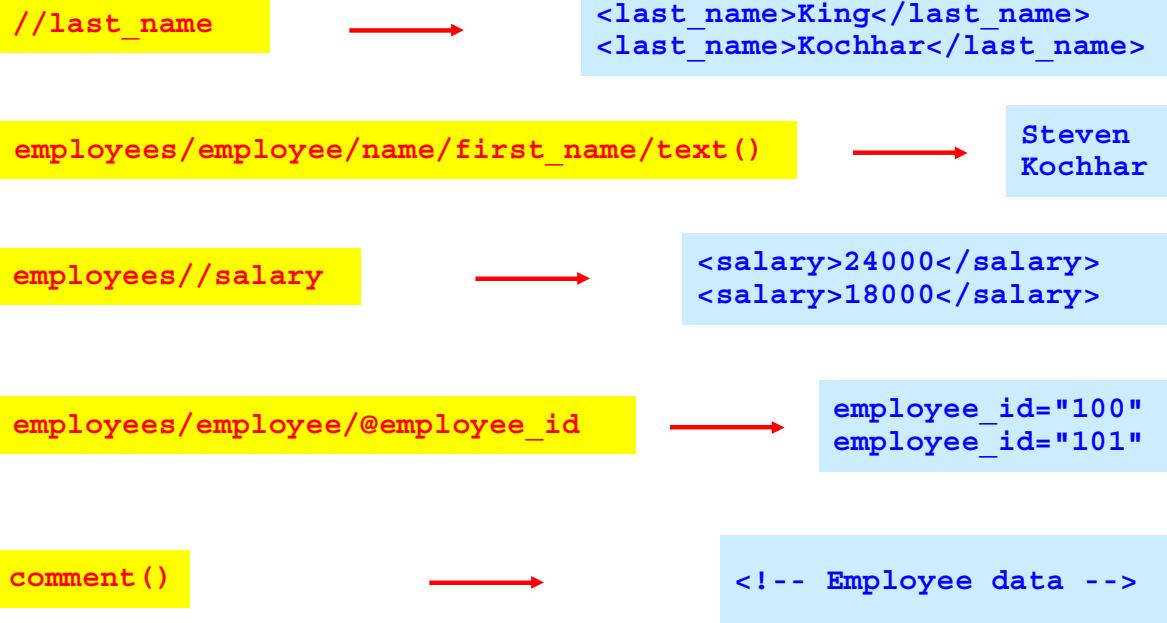
```
<?xml version="1.0"?>
<!-- Employee data --&gt;
&lt;employees&gt;
  &lt;employee employee_id="100"&gt;
    &lt;name&gt;
      &lt;first_name&gt;Steven&lt;/first_name&gt;
      &lt;last_name&gt;King&lt;/last_name&gt;
    &lt;/name&gt;
    &lt;salary&gt;24000&lt;/salary&gt;
  &lt;/employee&gt;
  &lt;employee employee_id="101"&gt;
    &lt;name&gt;
      &lt;first_name&gt;Neena&lt;/first_name&gt;
      &lt;last_name&gt;Kochhar&lt;/last_name&gt;
    &lt;/name&gt;
    &lt;salary&gt;18000&lt;/salary&gt;
  &lt;/employee&gt;
&lt;/employees&gt;</pre>
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The employees.xml document will be used in the XPath expressions in the next slide.

Abbreviated XPath Expression: Examples



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The examples in the slide are applied to the XML document on the previous slide, employees.xml.

Note: The context node is the document root; `<first_name>` and `<last_name>` are children of `<name>`; and the comment is a child of the document root.

Abbreviated XPath Expression Examples: The XML Document

```
<?xml version='1.0' encoding='UTF-8'?>
<departments>
  <department num="1">
    <department_id>10</department_id>
    <department_name>Administration</department_name>
    <manager_id>200</manager_id>
    <location_id>1700</location_id>
  </department>
  <department num="2">
    <department_id>20</department_id>
    <department_name>Marketing</department_name>
    <manager_id>201</manager_id>
    <location_id>1800</location_id>
  </department>
  <department num="3">
    <department_id>30</department_id>
    <department_name>Purchasing</department_name>
    <manager_id>114</manager_id>
    <location_id>1700</location_id>
  </department>
  <department num="4">
    <department_id>40</department_id>
    <department_name>Human Resources</department_name>
    <manager_id>203</manager_id>
    <location_id>2400</location_id>
  </department>
</departments>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Each XPath expression in the following slides is applied to the departments XML document in the slide.

XPath Predicates

- Is a Boolean expression in brackets that is evaluated for each node in the node-set

```
//department [department_name="Administration"]
```

- With a numeric result is converted to a Boolean by using the `position()` function

```
/departments/department [2]  
/departments/department [position ()=2]
```

- May be combined with logical operators

```
//department [@num>2 and @num<=4] /department_name
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A predicate in an XPath expression is a condition contained in brackets that appears after the node test of a location step. The predicate is evaluated as a Boolean expression for each node in a node-set. If the condition is true, the node is included; otherwise, the node is excluded from the results. Predicates, such as query expressions, filter the results.

- The first example returns the descendant `<department>` elements that contain a `<department_name>` element whose text node is Administration.
- The second example specifies a numeric value, which can be computed. Numeric results are converted to a Boolean expression by using an equality comparison of the value to the return value of the `position()` node-set function. The `position()` function determines the **proximity position** of a member node in a node-set. The proximity position is a numbered position of the node in the sequence in which it appears in a node-set. Proximity position numbers start at 1.
- The last example uses the `and` logical operator (in lowercase) to combine predicate expressions, which can be written using an implied `and` operation, as in this example:

```
//department [@num>2] [@num<=4] /department_name
```

You can combine predicates with an `or` logical operator (in lowercase), as in the following example: `//dcepartment[department_id="10" or (@num>1 and @num<4)]`

Note

The node-set results for each XPath expression are shown in the following examples:

- Expression: `//department [department_name="Administration"]`
`<department num="1">`
`<department_id>10</department_id>`
`<department_name>Administration</department_name>`
`<manager_id>200</manager_id>`
`<location_id>1700</location_id>`
`</department>`
- Expression: `/departments/department [2]`
`<department num="2">`
`<department_id>20</department_id>`
`<department_name>Marketing</department_name>`
`<manager_id>201</manager_id>`
`<location_id>1800</location_id>`
`</department>`
- Expression: `//department [@num>2 and @num<=4] /department_name`
`<department_name>Purchasing</department_name>`
`<department_name>Human Resources</department_name>`

XQuery: Review

- XQuery is the W3C standard language designed for querying (finding and extracting elements and attributes).
- XQuery is to XML what SQL is to database tables.
- ***XQuery is built on XPath expressions.***
- XQuery 1.0 and XPath 2.0 share the same data model and support the same functions and operators.
- By using XQuery you can query both structured and unstructured data:
 - Relational databases
 - XML documents
 - Other data sources with XML data view



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XQuery is the W3C standard language designed for querying, transforming, and accessing XML and relational data. XQuery is similar to SQL in many ways. Like SQL is designed for querying structured, relational data, XQuery is designed especially for querying unstructured, XML data from a variety of data sources.

You can use XQuery to query XML data wherever it is found, whether it is stored in the database tables, or available through Web services. In addition to querying XML data, you can use XQuery to construct XML data.

Oracle XML DB supports a native XQuery compilation engine that can parse and compile XQuery expressions into SQL-native structures for evaluation. This native execution significantly improves the performance of XQuery expressions in Oracle Database.

XQuery Terminology

- Nodes
 - Elements, attribute, text, namespace
 - Processing-instructions, comments, document nodes
- Family Relationship
 - Parent, children, siblings, ancestors, and descendants



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XQuery Review: books.xml Document Example

```
<?xml version="1.0" encoding="UTF-8"?>
<bookshop>
  <book category="CHILDREN">
    <title language="en">Love You Forever</title>
    <author>Robert Munsch</author>
    <year>1995</year>
    <price>4.98</price>
  </book>
  <book category="COMPUTERS">
    <title language="en">XML: Visual QuickStart Guide</title>
    <author>Kevin Howard Goldberg</author>
    <year>2008</year>
    <price>23.17</price>
  </book>
  <book category="COMPUTERS">
    <title language="en">Beginning XML, 5th Edition</title>
    <author>Joe Fawcett</author>
    <author>Liam R.E. Quin</author>
    <author>Danny Ayers</author>
    <year>2012</year>
    <price>25.97</price>
  </book>
</bookshop>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

FLWOR Expressions: Review

A FLWOR expression is one of the most important and powerful expressions in XQuery syntax. It supports iteration and binding of variables. It stands for:

- for
- let
- where
- order by
- return

Use the doc()
function to
open the
books.xml file

```
for $i in doc("books.xml")/bookshop/book
where $i/price > 20
order by $i/title
return $i/title
```

```
<?xml version="1.0" encoding="UTF-8"?>
<bookshop>
  <book category="CHILDREN">
    <title language="en">Love You Forever</title>
    <author>Robert Munsch</author>
    <year>1995</year>
    <price>4.98</price>
  </book>
  <book category="COMPUTERS">
    <title language="en">XML: Visual QuickStart Guide</title>
    <author>Kevin Howard Goldberg</author>
    <year>2008</year>
    <price>23.17</price>
  </book>
  <book category="COMPUTERS">
    <title language="en">Beginning XML, 5th Edition</title>
    <author>Elie Fréreault</author>
    <author>Liam R.E. Quin</author>
    <author>Danny Ayers</author>
    <year>2012</year>
    <price>25.97</price>
  </book>
</bookshop>
```

```
<title language="en">Beginning XML, 5th Edition</title>
→ <title language="en">XML: Visual QuickStart Guide</title>
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Expressions are the basic building blocks of XQuery. They are case-sensitive.

The acronym FLWOR (pronounced “flower”) represents the XQuery clauses for, let, where, order by, and return. A FLWOR expression has at least one for or let clause and a return clause; single where and order by clauses are optional.

- for: Analogous to the FROM clause of a SQL SELECT query. By using the for clause you can iterate across a range of sequence values and bind each one of them to one or more bind variables. At each iteration, the variables are bound in the order in which they appear.
- let: Analogous to the SQL SET statement. You can use the let clause to define variables and assign them, in turn, during iteration through a for clause. You can bind one or more variables. Just as with FOR, a variable can be bound by let to a value that is computed by using another variable that is listed previously in the binding list of let (or an enclosing for or let).
- where: Filters the for and let variable bindings according to a condition. This is similar to the SQL WHERE clause.
- order by: Arranges the result (returned by the WHERE clause) in ascending or descending order

Selecting Nodes From an XML Document

You can select nodes from an XML document by using:

- Functions
- Path expressions
- Predicates



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XQuery Basic Syntax Rules

- XQuery is case-sensitive.
- XQuery elements, attributes, and variables must be valid XML names.
- An XQuery string value can be in single or double quotes.
- An XQuery variable is defined with a \$ followed by a name, such as \$bookshop.
- XQuery comments are delimited by (: and :), for example, (: XQuery Comment :).



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

EXtensible Stylesheet Language (XSL) and XSL Transformations (XSLT)

- The W3C started to develop XSL because there was a need for an XML-based Stylesheet Language.
- XSL describes how the XML document should be displayed.
- XSLT transforms an XML document into plain text, HTML, or another XML document.
- XSLT uses XPath to navigate in XML documents.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Describe Extensible Markup Language (XML) and its structure and components
- Review Document Type Definition (DTD)
- Describe an XML Namespace
- Review and use an XML Schema
- Define and use the XML Path language (XPath)
- Define and use XQuery
- Review and use XSL Transformations (XSLT)



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 2-1: Overview

This practice covers the following topics:

- Review some fundamental concepts of XML



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The goal of this practice is to review some fundamental concepts of XML.

Practice 2-2: Overview

This practice covers the following topics:

- Access some of the available XML resources



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The goal of this practice is to access some of the available XML resources.

Introduction to Oracle XML DB

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

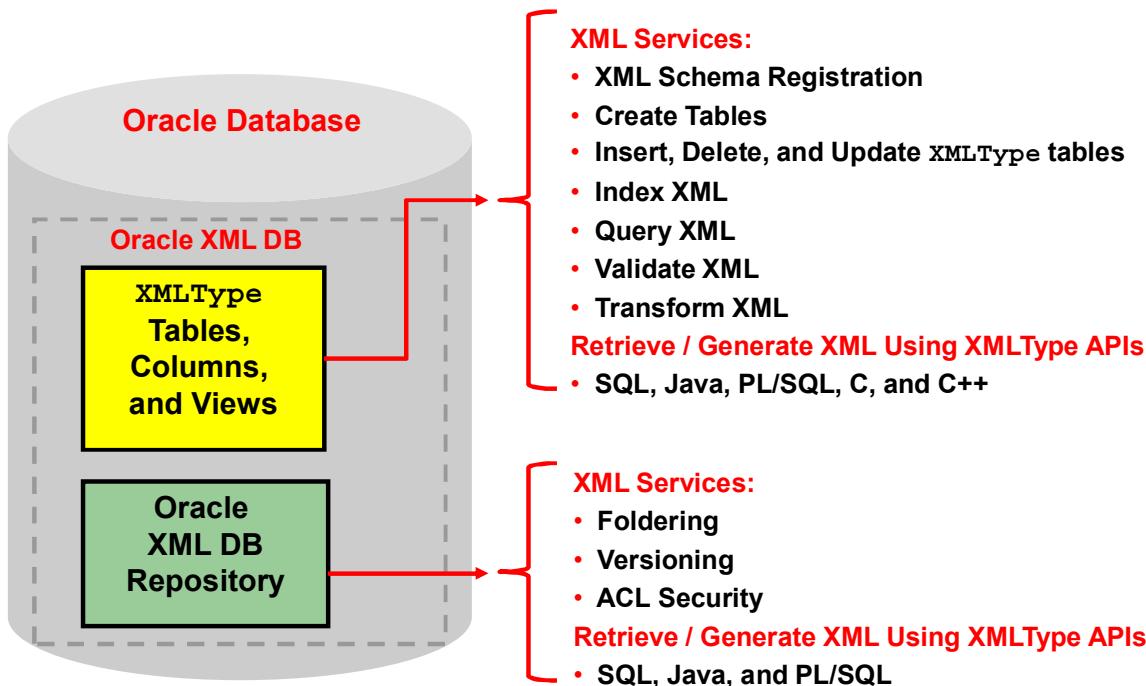
After completing this lesson, you should be able to:

- Define XML DB
- List the key features of Oracle XML DB



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle XML DB



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle XML DB is a set of Oracle Database technologies related to high-performance handling of XML data: storing, generating, accessing, searching, validating, transforming, evolving, and indexing. It provides native XML support by encompassing both the SQL and XML data models in an interoperable way. Oracle XML DB is included as part of Oracle Database. You learn about the key features that enable you to perform these operations in the lessons that follow.

Oracle XML DB Key Components

- **XMLType storage:** XML data, including XML schema definition files, can be stored in the XMLType columns and tables.
- **Oracle XML DB Repository:** This enables you to organize and manage database content in the form of files and folders, which are called *resources*. Oracle XML DB implements popular protocols such as FTP, HTTP, and Web-based Distributed Authoring and Versioning (WebDAV) to improve data access performance, and is designed around open standards and implemented to provide different views and intuitive access to a single repository. The XML Repository can be accessed by using Internet protocols such as WebDAV, FTP, and HTTP, and is as fast as accessing data from any file system.

Oracle XML DB: Benefits

The benefits of using Oracle XML DB are as follows:

- Unified relational data and XML content:
 - Enhances native database support for XML
 - Stores and manages structured, unstructured, and semi-structured data
 - Offers transparent XML and SQL interoperability
 - Exploits database features
 - Exploits XML features
- Faster storage and retrieval of complex XML documents:
 - Higher performance of XML operations
 - Higher scalability of XML operations



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Unifying Relational Data and XML Content with Oracle XML DB

Most application data and web content are stored in a relational database or a file system, or in a combination of both. When stored in relational storage, XML loses DOM fidelity. In industry, structured data is currently moving to XML for transport on the Internet to add structure to the content. Data is becoming more semi-structured. As the volume of XML data that is transported grows, the cost of regenerating these XML documents grows, and these storage methods become less effective for accommodating XML content. With Oracle XML DB, you have enhanced native support for XML.

With Oracle XML DB, you can store and manage structured, unstructured, and pseudo- or semi-structured data by using a standard data model and standard SQL and XML. You can store structured data in structured storage such as tables and columns, and unstructured data in large objects (LOBs). Oracle Database 11g introduces binary XML storage.

By using Oracle XML DB, you have complete transparency and interchangeability between XML and SQL. With Oracle XML DB, you can perform:

- XML operations on structured data (such as tables)
- SQL operations on XML documents

Oracle Database has the following key database capabilities for working with XML:

- **Indexing and search:** Applications use queries such as "find all the product definitions created between March and April 2002", a query that is typically supported by a B-tree index on a date column. Oracle XML DB can enable efficient structured searches on XML data, saving content-management vendors the need to build proprietary query APIs to handle such queries.
- **Updates and transaction processing:** Fast updates of subparts of records, with minimal contention between users trying to update. As traditionally document-centric data participate in collaborative environments through XML, this requirement becomes more important. File or CLOB storage cannot provide the granular concurrency control that Oracle XML DB does.
- **Managing relationships:** Data with any structure typically has foreign key constraints. XML data stores generally lack this feature, so you must implement any constraints in application code. With Oracle XML DB, you can constrain XML data according to XML schema definitions, and hence achieve control over relationships that structured data has always enjoyed.
- **Multiple views of data:** Most enterprise applications need to group data together in different ways for different modules. This is why relational views are necessary: to allow for these multiple ways to combine data. By allowing views on XML, Oracle XML DB creates different logical abstractions on XML (for example, for consumption by different types of applications).
- **Performance and scalability:** Users expect data storage, retrieval, and query to be fast. Loading a file or CLOB value, and parsing, are typically slower than relational data access. Oracle XML DB dramatically speeds up XML storage and retrieval.
- **Ease of development:** Databases are foremost an application platform that provides standard, easy ways to manipulate, transform, and modify individual data elements. While typical XML parsers give standard read access to XML data, they do not provide an easy way to modify and store individual XML elements. Oracle XML DB supports several standard ways to store, modify, and retrieve data. These include XML Schema, XQuery, XPath, DOM, and Java.

If the drawbacks of XML file storage force you to break down XML into database tables and columns, there are several XML advantages that you still have:

- **Structure independence:** The open content model of XML cannot be captured easily in the pure tables-and-columns world. XML schemas allow global element declarations, not just scoped to a container. Hence, you can find a particular data item regardless of where in the XML document it moves to as your application evolves.
- **Storage independence:** When you use relational design, your client programs must know where your data is stored, in what format, what table, and what the relationships are among those tables. By using XMLType, you can write applications without that knowledge, and database administrators can map structured data to physical table-and-column storage.

- **Ease of presentation:** XML is understood natively by web browsers, many popular desktop applications, and most Internet applications. Relational data is generally not accessible directly from applications. Additional programming is required to make relational data accessible to standard clients. Oracle XML DB stores data as XML and makes it available as XML outside the database. No extra programming is required to display database content.
- **Ease of interchange:** XML is the language of choice in business-to-business (B2B) data exchange. If you are forced to store XML in an arbitrary table structure, you are using some kind of proprietary translation. Whenever you translate a language, information is lost and interchange suffers. By natively understanding XML and providing DOM fidelity in the storage/retrieval process, Oracle XML DB affords a clean interchange.

Users today face a performance barrier when storing and retrieving complex, large, or many XML documents. Oracle XML DB provides high performance and scalability for XML operations. The major performance features are Native XMLType, XQuery, XPath, and XSLT support, indexing, both full-text and XML, and a hierarchical index over Oracle XML DB Repository.

Oracle XML DB: Features

- XMLType Data Type
- XML Schema Support
- XMLType Storage Models
- XML/SQL Duality
- SQL/XML Standard Functions
- Automatic Rewriting of XQuery and XPath Expressions
- Oracle XML DB Repository



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLType Data Type

XMLType is an abstract native SQL data type for XML data. The term native refers to the natural way of storing xml data which is more efficient than storing it as a serialized text. Native is an internal data type aware representation. XMLType provides methods that allow operations such as XML schema validation and XSL transformation of XML content. You can use XMLType as you would any other SQL data type. For example, you can use XMLType when you do any of the following:

- Create a column in a relational table
- Declare a PL/SQL variable
- Define or call a PL/SQL procedure or function

XML Schema Support

Support for the Worldwide Web Consortium (W3C) XML Schema Recommendation is a key feature in Oracle XML DB. XML Schema specifies the structure, content, and certain semantics of XML documents. It is described in detail at <http://www.w3.org/TR/xmlschema-0/>.

The W3C Schema Working Group publishes a particular XML schema, often referred to as the schema for schemas, that provides the definition, or vocabulary, of the XML Schema language. An XML schema definition (XSD), also called an XML schema, is an XML document that is compliant with the vocabulary defined by the schema for schemas.

An XML schema uses vocabulary defined by the schema for schemas to create a collection of XML Schema type definitions and element declarations that comprise a vocabulary for describing the contents and structure of a new class of XML documents, the XML instance documents that conform to that XML schema.

Note

In this course, we use the term "XML schema" (lower-case "s") to reference any XML schema that conforms to the W3C XML Schema (upper-case "S") Recommendation. Since an XML schema is used to define a class of XML documents, the term "instance document" is often used to describe an XML document that conforms to a particular XML schema.

XMLType Storage Models

XMLType is an abstract data type that provides different storage models to best fit your data and your use of it. As an abstract data type, your applications and database queries gain flexibility: the same interface is available for all XMLType operations. Because different storage (persistence) models are available, you can customize performance and functionality to best fit the kind of XML data that you have and the pattern of its use. One key decision to make when using Oracle XML DB for persisting XML data as XMLType is, thus, which storage model to use for which XML data.

XML/SQL Duality

A key objective of Oracle XML DB is to provide XML/SQL duality. XML programmers can leverage the power of the relational model when working with XML content, and SQL programmers can leverage the flexibility of XML when working with relational content. Thereby, you can use the most appropriate tools for a particular business problem.

XML/SQL duality means that the same data can be exposed as rows in a table and manipulated using SQL, or exposed as nodes in an XML document and manipulated using techniques such as DOM and XSL transformation. Access and processing techniques are independent of the underlying storage format.

These features provide simple solutions to common business problems. For example:

You can use Oracle XML DB SQL functions to generate XML data directly from a SQL query. You can then transform the XML data into other formats, such as HTML, by using the database-resident XSLT processor.

You can access XML content without converting between different data formats, using SQL queries, online analytical processing (OLAP), and business-intelligence/data warehousing operations.

You can perform text, spatial data, and multimedia operations on XML content.

SQL/XML Standard Functions

Oracle XML DB provides the SQL functions that are defined in the SQL/XML standard.

SQL/XML functions fall into two groups:

- Functions that you can use to generate XML data from the result of a SQL query. In this course, these are called SQL/XML publishing functions. They are also sometimes called SQL/XML generation functions.
- Functions that you can use to query and access XML content as part of normal SQL operations. In this course, these are called SQL/XML query and access functions.

Using SQL/XML functions, you can address XML content in any part of a SQL statement. These functions use XQuery or XPath expressions to traverse the XML structure and identify the nodes on which to operate. The ability to embed XQuery and XPath expressions in SQL statements greatly simplifies XML access.

Automatic Rewriting of XQuery and XPath Expressions

SQL/XML functions and `XMLType` methods use XQuery or XPath expressions to search collections of XML documents and to access a subset of the nodes contained within an XML document. In many cases, Oracle XML DB is able to automatically rewrite such expressions to code that executes directly against the underlying database objects.

Overview of Oracle XML DB Repository

Oracle XML DB Repository is a component of Oracle Database with which you can handle XML data using a file, folder, or URL metaphor. The repository contains resources, which can be either folders (directories, containers) or files.

Summary

In this lesson, you should have learned how to:

- Define XML DB
- List the key features of Oracle XML DB



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Storing XML Data in Oracle XML DB

4

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe XMLType data type
- Create XMLType objects
- Describe the XML schema support in Oracle XML DB
- Describe XMLType storage options
- Load data into XMLType
- Specify SQL constraints on XMLType tables



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In this lesson, you should learn about:

- The XMLType data type, which is an integral part of the Oracle XML database environment
- Creating XMLType columns and XMLType tables
- The XML schema support in Oracle XML DB: Registering an XML schema and deleting a registered XML schema in Oracle XML DB
- This lesson covers the two types of storage models available in Oracle XML DB:
 - Object-relational
 - Binary XML
- You also learn how to load data into XMLType tables and how to specify SQL constraints on XMLType tables.

Lesson Agenda

- Using XMLType
- Choosing an XMLType storage model
 - Binary XML storage
 - Object-relational storage
- Loading data into XMLType
- Specifying SQL constraints



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLType: Overview

XMLType:

- Is an abstract data type for native handling of XML data in the database
- Can be used as the data type for a column or object table
- Stores XML content
 - Can be used in PL/SQL stored procedures as parameters, return values, and variables
 - Can represent an XML document in the database, so it is accessible in SQL
- Provides a SQL API with functions that operate on XML data



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLType:

- Is a native data type that stores XML content. Native indicates that it is the natural way of storing XML and it is more efficient than storing it as a serialized text.
- Can be used as the data type of a column in a table or view, or as the data type of an object table
- Designed to store XML content in an optimal way
- Can be used in PL/SQL stored procedures as parameters, return values, and variables
- Can represent an XML document in the database, so it is accessible in SQL
- Provides a SQL API framework that has built-in functions that operate on XML content. For example, you can use XMLType functions to create, extract, and index XML data that is stored in the Oracle Database.
- Provides its functionality through a set of APIs that are available to PL/SQL, Java, C++, and C

In Oracle XML DB, you can store an XML document in two ways: as an XMLType column in a relational table, or as an XML object in an XMLType table.

With the capabilities of XMLType, you can leverage the power of the relational database while working in the context of XML. Similarly, you can leverage the power of XML standards while working in the context of a relational database.

Using XMLType

Use XMLType to:

- Store and query XML data
- Provide efficient XPath access
- Shield applications from storage models
- Prepare for future optimizations



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use XMLType whenever you want to use the database for persistent storage of XML data.

Use XMLType to use SQL queries on a part of the document or on the entire document.

XMLType features include strong typing inside SQL statements and PL/SQL functions. Strong typing ensures that the values passed are XML values and not arbitrary text strings.

The XML schema language defines 47 scalar data types. This provides for strong typing of elements and attributes. Oracle XML DB leverages this strong typing feature of XML schema to process XML data safely and efficiently.

XMLType uses the built-in C XML parser and processor. Therefore, it provides better performance and scalability when used inside the server.

Because Oracle Database is natively aware that XMLType can store XML data, better optimizations and indexing techniques are possible. By writing applications to use XMLType, you can achieve these optimizations and enhancements, and preserve them in future releases without needing to rewrite the applications. By using XMLType instead of the standard relational or binary XML storage, applications can accommodate various storage alternatives while minimizing the changes to the SQL statements in the application.

Declaring an XMLType Table, Column, and Attribute: Examples

- For a column in a table:

```
CREATE TABLE emp_resumes (
    employee_id NUMBER(6) PRIMARY KEY,
    resume      XMLType);
```

1

- For a table:

```
CREATE TABLE emp_xmlresumes OF XMLType;
```

2

- For an attribute in an object type:

```
CREATE TYPE resume_t AS OBJECT (
    employee_id NUMBER(6),
    resume      XMLType);
```

3

ORACLE

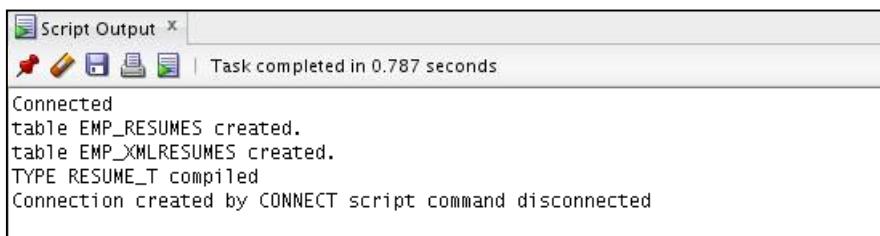
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The three ways in which XMLType data type can be used are shown in the slide:

- As a resume column type in a relational table called emp_resumes
- As the data type for objects in the object table called emp_xmlresumes
- As the data type for the resume attribute in the resume_t object type definition

Note: You can also use the XMLType data type when defining views. If you create an XMLType view, or a relational view that includes the XMLType column, you can use Oracle XML DB to expose the content stored in relational tables and external data sources as XML documents. You learn how to create an XMLType view in the lesson titled “Transforming and Validating XMLType Data.”

The following is the output of the script that contains the three code examples:



```
Script Output x
Task completed in 0.787 seconds
Connected
table EMP_RESUMES created.
table EMP_XMLRESUMES created.
TYPE RESUME_T compiled
Connection created by CONNECT script command disconnected
```

XMLType Storage Models

Oracle XML DB provides two storage models for XMLType tables and columns:

- **Binary XML storage:**
 - XMLType data is stored in a post-parse, binary format specifically designed for XML data.
 - Binary XML is compact, post-parse, XML schema-aware XML. This is also referred to as post-parse persistence.
- **Object-relational storage:**
 - XMLType data is stored as a set of objects.
 - This is also referred to as structured storage and object-based persistence.
 - You can embed CLOB storage within object-relational storage. This is called **hybrid** storage.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLType is a native server data type that enables the database to understand that a column or table contains XML data. This is similar to the DATE data type enabling the Oracle database to understand that a column contains a date value. XMLType is an abstract data type that provides different storage models to best fit your data and your use of it. As an abstract data type, your applications and database queries gain flexibility, because the same interface is available for all XMLType operations. You can optimize the performance and functionality by selecting a storage model that best fits the kind of XML data that you have and the pattern of its use. The purpose of abstraction in XMLType is to have a single data type from your application's point of view.

XMLType tables and columns can be stored in the following ways:

- **Binary XML Storage:** The XMLTYPE data is stored in post-parsed binary format specifically designed for XML data. Binary XML is compact, post-parsed, and XML schema-aware XML. The biggest advantage of Binary XML storage is its flexibility. You can use it for schemaless documents, or when the schema allows for high availability. This is also referred to as post-parse persistence.

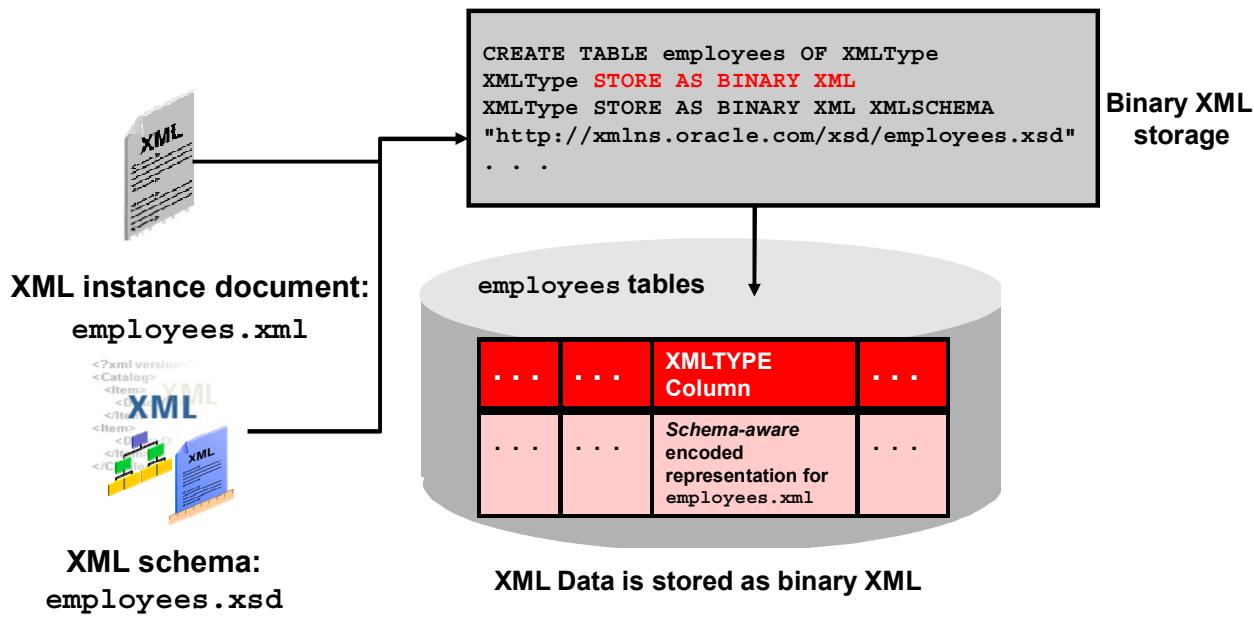
Object-Relational Storage: The XMLTYPE data is stored relationally as a set of objects. This is also referred to as structured storage and object-based persistence. The biggest advantage of object-relational storage is that it provides the best performance in structured cases. The query performance matches that of relational tables, and updates can be performed in-place. It also provides relational-like schema evolution capability.

In object-relational XMLType storage, an XML document is broken up and stored object-relationally, but you can choose to store one or more of its XML fragments as embedded CLOB instances. A typical use case for this is mapping an XML-schema complexType or a complex element to CLOB storage, because you generally access the entire fragment as a unit. This is sometimes called *hybrid* storage. For example, you can embed CLOB storage within o-r storage.

Note

- Each of the storage models has its own advantages and disadvantages in different dimensions, such as performance and flexibility. No single storage model is right for all use cases.
- For additional information about the available XMLType storage models, see the following resources:
 - The *Oracle XML DB Developer's Guide 12c Release 1 (12.1)* documentation reference
 - The *Oracle XML DB: Choosing the Best XMLType Storage Option for Your Use Case* white paper on OTN:
<http://www.oracle.com/technetwork/database/features/xmldb/xmlchoosestorage-v1-132078.pdf>

XMLType Storage: Binary XML for Schema-Based Storage



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle Database 11g introduced a new storage model known as binary XML storage. The binary XML format is the new option for storing and manipulating XML data within the database. This format is also used for transferring XML documents to and from clients and application tiers.

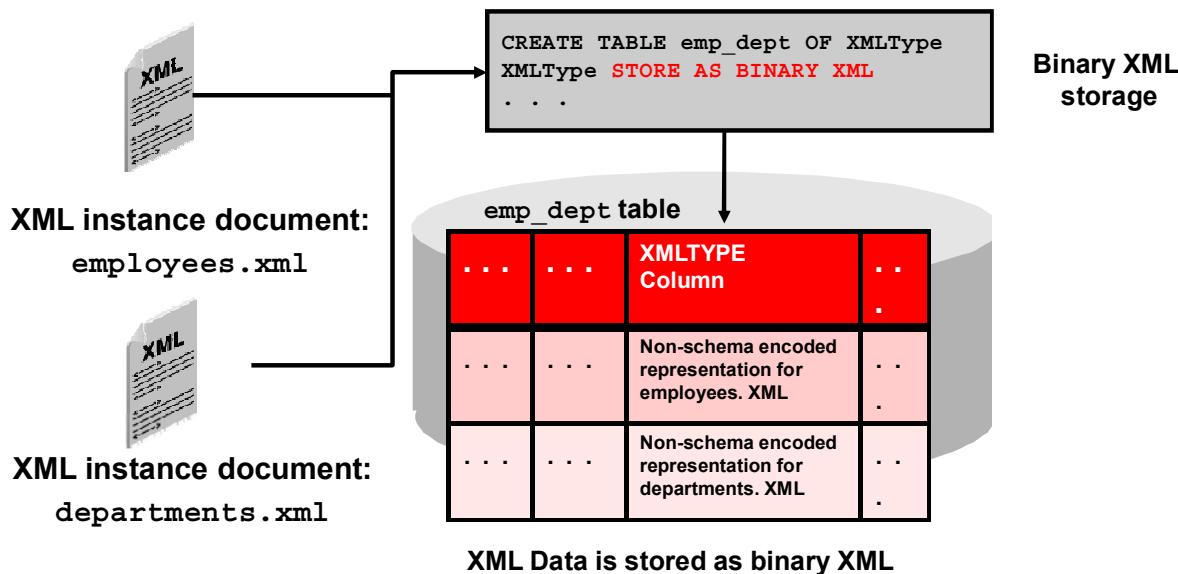
The binary XML storage option is XML schema-aware, thereby allowing better disk space efficiency and query performance.

Binary XML storage allows for very complex and variable data, which, in the o-r storage model, could necessitate the use of many database tables.

Binary XML storage is the closest thing to a universal storage model for XML data. You can use binary XML data for a very wide range of use cases, from document-centric to data-centric.

The slide example shows how Oracle XML DB creates XML schema-based XMLType tables using an XML document and a mapping specified in an XML schema using the binary XML storage model.

XMLType Storage: Binary XML for Non-Schema-Based Storage



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A non-schema-based storage is preferred if you want to store a mixed set of structured or unstructured documents in an `XMLType` table or column.

Binary XML: Advantages

Binary XML provides:

- Reduced storage requirements
- Efficient leaf- and fragment-level extraction
- More efficient CPU, network, and memory utilization
- Flexible XML schema use
- Streaming XML parsing and validation
- Support for natural language-sensitive operations



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

With binary XML, you save memory space, network bandwidth, and processing time. The following are some of the advantages of binary XML:

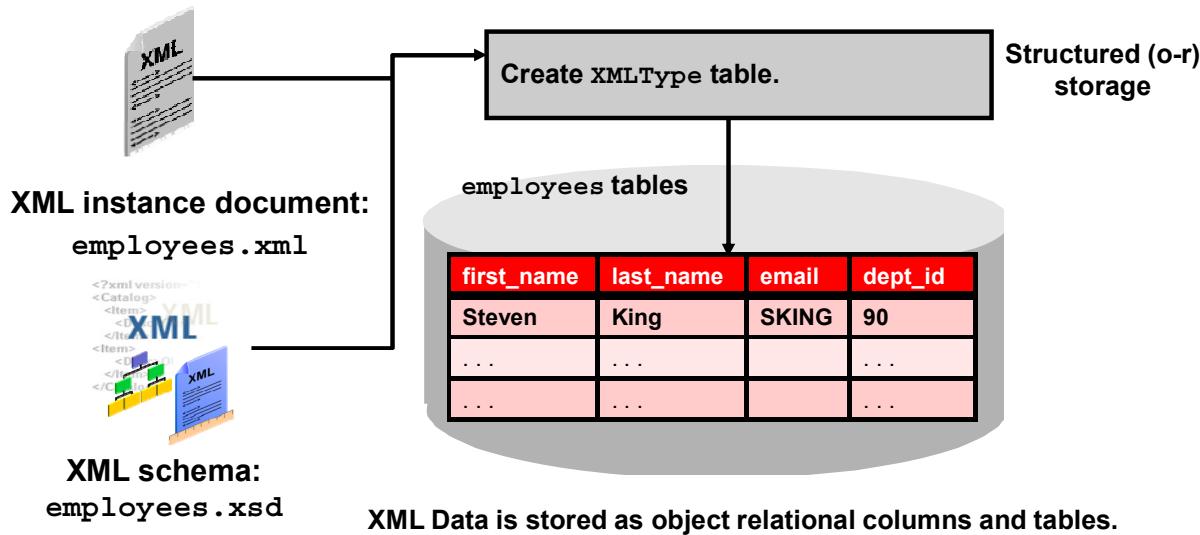
- **Reduced storage requirements:** Oracle binary XML is a compact representation of an XML document. This significantly reduces the amount of disk space required to store XML documents.
- **Efficient leaf- and fragment-level extraction:** The binary XML format supports streaming XPath evaluation, allowing multiple leaf nodes or fragments to be extracted in a single pass through an XML document.
- **More efficient CPU, network, and memory utilization:** XML parsing and serialization of XML are time consuming. Oracle binary XML addresses this problem. With binary XML, the internal representation of XML is the same, regardless of whether the XML is on disk or in memory. Because the same representation of XML is shared by all application tiers, this enables efficient exchange of XML content.
- **Flexible XML schema support:** Oracle binary XML supports both XML schema-based and non-schema-based data. The binary XML model allows XML documents that are associated with multiple XML schemas to be stored in the same table or column.

- **Streaming XML validation:** With the binary XML storage option, XML schema validation is performed with a new streaming implementation. Streaming XML schema validation requires less memory. Therefore, it performs very efficiently and scales well when validating a large number of XML documents or large-sized XML documents. Binary XML also reduces the overhead needed to validate XML documents during update operations by allowing fragment-level rather than document-level validation. This means that only those parts of an XML document that are modified need to be revalidated following an update.
- **Support for natural language-sensitive operations:** Oracle binary XML supports the W3C XLIFF standard, allowing content to be managed in a natural language-sensitive manner. If a document contains language-sensitive content, and if you access the document, only content that is appropriate to the current language is returned.

Note: XLIFF is the acronym for XML Localization Interchange File Format.

XMLType Storage: Structured (object-relational) Storage

A schema-based XMLType document can be stored as a set of SQL objects.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Object-relational (o-r) storage requires a registered XML schema. With o-r or structured storage, Oracle XML DB decomposes the content of an XML document and stores it as a set of SQL objects rather than simply storing the document as text in a LOB. Oracle XML DB creates the XML schema-based XMLType tables by using an XML document and a mapping specified in an XML schema.

The graphic in the slide shows the creation of an XMLType table, where the contents of XMLType are constrained to a global element that is defined by a registered XML schema and the contents of XMLType are stored by using a set of SQL objects.

A SQL type definition is generated from each complexType defined by the XML schema. Each element or attribute defined by complexType becomes a SQL attribute in the corresponding SQL type. Oracle XML DB automatically maps the 47 scalar data types defined by the W3C XML Schema Recommendation to the 19 scalar data types supported by SQL. For details about mapping complexType to a SQL type definition, refer to the *Oracle XML DB Developer's Guide 12c Release 1 (12.1)*.

In the slide example, employees.xsd contains four elements: `first_name`, `last_name`, `email`, and `dept_id`. When you create the XMLType table, those elements are mapped into four columns in the employees table. If one of the elements in the XML schema document is a collection such as `lineitems`, then that element is stored in a separate table from the earlier four elements.

The generated SQL types enable XML content (compliant with the XML schema) to be decomposed and stored in the database as a set of objects without loss of information.

When the document is inserted, the constructs defined by the XML schema are mapped directly to the equivalent SQL types. This enables Oracle XML DB to leverage the full power of the Oracle database when managing XML, and can lead to significant reductions in the amount of space required to store the document. It can also reduce the amount of memory required to query and update the XML content.

XMLType Structured (Object-Relational) Storage: Advantages and Disadvantages

- Advantages:
 - Near-relational query and update performance
 - Optimized memory management
 - Reduced storage requirements
 - B-tree indexing
 - In-place updates
- Disadvantages:
 - Increased processing overhead during ingestion and full retrieval of XML data
 - Reduced flexibility in the structure of the XML that can be managed by a given XMLType table or column.
 - Difficulty in varying instance structures



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Object-relational storage advantages over the other storage models include near-relational query and update performance, optimized memory management, reduced storage requirements, B-tree indexing, and in-place updates. These advantages are at the cost of increased processing overhead during ingestion and full retrieval of XML data, and reduced flexibility in the structure of the XML that can be managed by a given XMLType table or column. Structural flexibility is reduced because data and metadata (such as column names) are separated in object-relational storage. Instance structures cannot vary easily. Structured storage is particularly appropriate for highly structured data whose structure does not vary, if this maps to a manageable number of database tables and joins.

Specifying Binary XML Storage

- Use STORE AS BINARYXML during table creation to specify binary XML storage. This is the default.
- When you create an XMLType table or column and specify binary XML storage, you have the following options:
 - Encoding the column or table data as non-schema-based binary XML
 - Encoding the column or table data to conform to a single XML schema
 - Encoding the column or table data to conform to whichever XML schema it references



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If you use the STORE AS BINARYXML clause and specify an XML schema that the XML documents must conform to, you can use that XML schema to create only XMLType tables and columns that are stored as binary XML. Note that you cannot use the same XML schema to create XMLType tables and columns that are stored object-relationally. If you use a storage model other than binary XML for the registered XML schema, you can use that XML schema to create only XMLType tables and columns that are not stored object-relationally.

When you create an XMLType table or column and specify binary XML storage, you can also specify how to encode the column or table to make use of the XML schemas. You can use any of the three options mentioned in the slide.

When you specify that all documents must conform to the same XML schema, as an option, you can specify that non-schema-based documents can also be stored in the same column. You can use the option to encode the column or table data to conform to whichever XML schema it references. In this case, you can choose to specify that each document can reference any XML schema, and that XML schema is used to encode that particular XML document. However, as an option, you can specify that non-schema-based documents can also be stored in the same column. You can specify multiple versions of the same XML schema (that is, store documents that conform to different versions). Each document encodes according to the schema it references.

Specifying Binary XML Storage: Examples

```
-- Example 1
CREATE TABLE po_tab OF XMLType
XMLType STORE AS BINARY XML;

-- Run code_04_18_n before running this script.

-- Example 2
CREATE TABLE po_tab_csx_table OF XMLType
XMLType STORE AS BINARY XML
XMLSCHEMA
"http://xmlns.oracle.com/xsd/po_bin.xsd"
ELEMENT"PurchaseOrder";

-- Example 3
CREATE TABLE multi_schema_table OF XMLType
XMLType STORE AS BINARY XML
ALLOW ANYSCHEMA;
```

1

2

3



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Example 1 shows how to create an XMLType table based on binary storage.

Example 2 shows that the contents of XMLType are constrained to a single global element defined by a registered XML schema. In this case, the XML documents are stored by using the XML schema-based binary XML format. You cannot insert an XML document that does not conform to the specified XML schema.

To create an XMLType column conforming to multiple XML schemas, you can use the ALLOW ANYSCHEMA option. Example 3 shows how to create an XMLType column that conforms to multiple XML schemas. In this case, any document that conforms to a registered XML schema is encoded by using that XML schema. The following is another example that shows how to specify the storage clause:

```
CREATE TABLE po_tab_as_column(id NUMBER, xml_document
XMLType) XMLType COLUMN xml_document
STORE AS BINARY XML(TABLESPACE USERS STORAGE(INITIAL 4K NEXT 32K));
```

```
drop table po_tab_as_column succeeded.
CREATE TABLE succeeded.
```

Registering an XML Schema for Binary XML

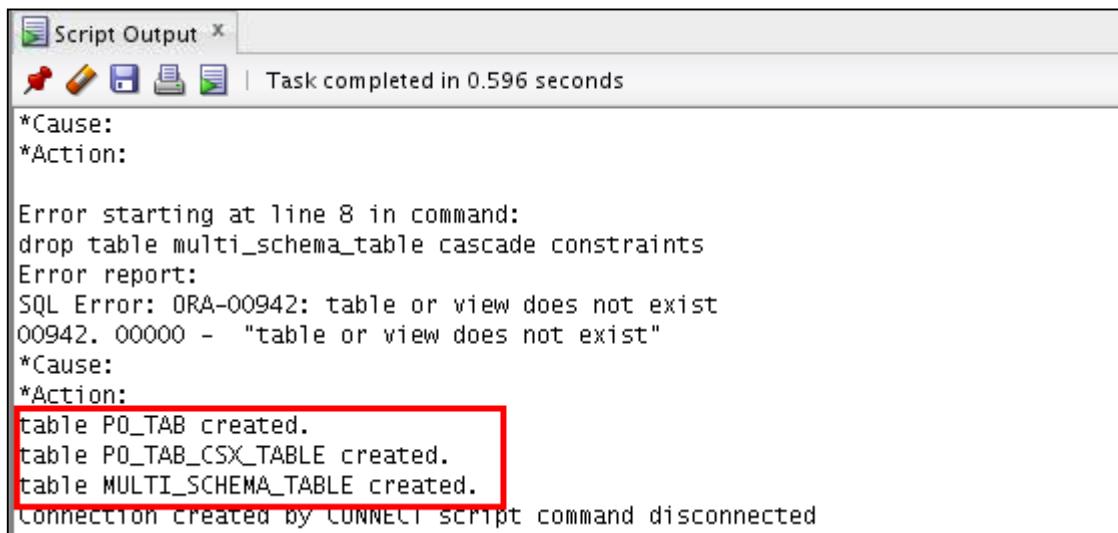
The second code example on the previous page requires that you register the XML schema with the URL specified as `http://xmlns.oracle.com/xsd/po_bin.xsd` before you can successfully run the code. You can use FTP to load the `po_bin.xsd` schema definition document into the database before executing the PL/SQL procedure. The most important option is `DBMS_XMLSHEMA.REGISTER_BINARYXML`, which indicates that the XML schema is used for binary XML storage.

```
BEGIN  
  DBMS_XMLSHEMA.registerSchema (  
    SCHEMAURL =>  
      'http://xmlns.oracle.com/xsd/po_bin.xsd',  
    SCHEMADOC =>  
      xdbURITYPE ('/home/OE/xsd/po_bin.xsd').getClob(),  
    local => TRUE,  
    genTypes => FALSE,  
    genTables => FALSE,  
    force => FALSE,  
    OPTIONS => DBMS_XMLSHEMA.REGISTER_BINARYXML
```

The output of running the prior code example is as follows:

```
*Cause: The given schema URL does not refe  
*Action: Make sure the specified schema has  
Connected  
anonymous block completed  
Connection created by CONNECT script command
```

The output of running the code example script that contains the three examples in the previous slide is as follows:



The screenshot shows the Oracle SQL Developer interface with the "Script Output" tab selected. The output window displays the results of the executed PL/SQL script. The output is as follows:

```
Script Output X  
Task completed in 0.596 seconds  
*Cause:  
*Action:  
  
Error starting at line 8 in command:  
drop table multi_schema_table cascade constraints  
Error report:  
SQL Error: ORA-00942: table or view does not exist  
00942. 00000 - "table or view does not exist"  
*Cause:  
*Action:  
table PO_TAB created.  
table PO_TAB_CSX_TABLE created.  
table MULTI_SCHEMA_TABLE created.  
Connection created by CONNECT script command disconnected
```

You can see the objects created by the `registerSchema` function by writing a query that uses the `USER_OBJECTS` data dictionary view.

Note: You do not want to create default tables for most global elements. Elements that never serve as the root element for an XML instance document do not need default tables. Creating default tables can lead to significant overhead in processor time and space used, especially if an XML schema contains a large number of global element definitions. In general, you want to prevent the creation of a default table for any global element that you are sure will not be used as a root element in any document.

Allowing Non-Schema-Based Storage

Use the ALLOW NONSCHEMA clause to specify non-schema-based document storage.

```
CREATE TABLE multi_schema_tab1 OF XMLType  
XMLType STORE AS BINARY XML  
XMLSCHEMA  
"http://xmlns.oracle.com/xsd/po_bin.xsd"  
ELEMENT "PurchaseOrder"  
ALLOW NONSCHEMA;
```

```
table MULTI_SCHEMA_TAB1 created.  
Connection created by CONNECT script command disconnected
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can specify whether non-schema-based documents can be stored in an XMLType table for binary XML. The slide example shows the creation of an XMLType table for binary XML. The STORE AS BINARY XML clause is not required since this is the default storage model.

The multi_schema_tab1 table stores non-schema-based XML documents as well.

Following is a summary of some of the possibilities of using the CREATE TABLE options for binary XML:

- STORE AS BINARY XML. This is the default.
- STORE AS BINARY XML XMLSCHEMA ...
- STORE AS BINARY XML XMLSCHEMA ... ALLOW NONSCHEMA
- STORE AS BINARY XML ... ALLOW ANYSCHEMA
- STORE AS BINARY XML ... ALLOW ANYSCHEMA ALLOW NONSCHEMA

Note

- In the absence of the XMLSCHEMA keyword, encoding is for non-schema-based documents.
- In the absence of the ALLOW NONSCHEMA keyword, but where the XMLSCHEMA keyword is present, encoding is for the single XML schema that is specified.
- In the absence of the ALLOW NONSCHEMA keyword, but where the ALLOW ANYSCHEMA keyword is present, encoding is for any XML schema that is referenced.

XMLType Storage: Use Case Scenarios

1. An XML store with very little query or update requirements
2. Extraction-Transformation-Loading (ETL): XML used only as a staging area
3. XML persistence requiring interoperability with relational systems (including updates)



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide shows some typical use scenarios of Oracle XML DB. The following explains the seven use case scenarios:

Extraction-Transformation-Loading (ETL): XML is used only as a staging area

This refers to instances where you need to persist XML in the database before transforming it into your operational systems (mainly relational). The XML data in this case is highly structured and conforms to an XML schema. In this use case, you usually produce relational values from XML as well as generating XML from relational data.

XML persistence requiring interoperability with relational systems (including updates).

This is similar to use case 2, except that the XMLType data requires updates. The updates could update partial XML data, known as piecewise updates. Store your XML data using XMLType object-relational storage. This model has excellent support for piecewise updates.

Note: For additional information about the XMLType storage use case scenarios, see the *Oracle XML DB Home page at <http://www.oracle.com/technetwork/database-features/xmldb/overview/index.html>.*

XMLType Storage: Use Case Scenarios

4. Semi-structured XML persistence that includes updates.
 - Store the XML data using binary XML.
 - Create relational views over XML using the XMLTable function.
 - Queries should be written against the relational views.
5. Business Intelligence (BI).
 - Use binary XML storage with structured XMLIndex on it.
 - Create relational views over XML using the XMLTable function.
 - Write queries against relational views.
6. XML content repository with full text searches:
 - Use binary XML storage.
 - Create context index on the binary XMLType column.
7. Data integration from diverse data sources to allow a uniform query interface:
 - Use binary XML storage.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Semi-structured XML persistence that includes updates

In this case, either the xml schema is variable, or large portions of the xml schema are not well defined. The use of XMLType o-r storage is not feasible:

- Store your XML data using binary XML.
- Create relational views over XML using the XMLTable function, where the views project all columns of interest to the BI application.
- Queries should be written against the relational views.

Business Intelligence (BI)

SQL constructs such as order-by, group-by, and window enable powerful BI queries over relational data. The XMLTable function allows values in XML to be projected out as a virtual table. Order-by, group-by, and window constructs can operate on columns of the virtual table.

- Binary XML is the ideal storage option.
- Create relational views over XML using the XMLTable function, where the views project all columns of interest to the BI application.
- Queries should be written against the relational views.

XML content repository with full text searches

If your application needs to do a full text search on the XML data, use binary XML storage with Oracle Text index on it.

Data integration from diverse data sources to allow a uniform query interface

If your XML data comes from several different data sources, each having its own schema, then you should store it in the binary XML format.

Choosing an XMLType Storage Model: Data Centric



	Data-Centric	Document-Centric		
Use Case	XML schema-based data, with little variation and little structural change over time	Variable, free-form data, with some fixed embedded structures	Variable, free-form data	
Typical Data	Employee record	Technical article with author, date, and title fields	Web document or book chapter	
Storage Model	Object-Relational (Structured)	Binary XML		
Indexing	B-tree index	<ul style="list-style-type: none"> • XMLIndex with structured and unstructured components • XML Full-Text Index 	<ul style="list-style-type: none"> • XMLIndex with unstructured component index • XML Full-Text Index 	

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLType has different storage models, and some models can be configured in multiple ways. Each model has its advantages, depending on the context.

The first thing to consider when you choose an XMLType storage model is the nature of your XML data and the ways in which you use the data. You should have an understanding of whether your use case is primarily data-centric or document-centric.

Document-centric use of XML data means that you always use the data in its complete form. If you want to use the data, you always retrieve it in its total form.

In the case of **data-centric** use of XML data, the focus is only on pieces of the total set of XML data within a document but not on the complete document. You can decide the storage model depending on how data-centric or document-centric your data is.

Data-centric

- Data is, in general, highly structured with a relatively static and predictable structure.
- Applications take advantage of the structure.
- Data conforms to an XML schema.

After identifying how data-centric or document-centric your data is, you can choose the type of XML storage model. The graphic in the slide shows a spectrum from totally structured to totally unstructured data along with examples. You can identify whether your use case is at the end of the spectrum or closer to the middle. That is, you can identify how data-centric or document-centric your use case is.

- Use object-relational (structured) storage with B-tree indexing for purely data-centric cases. For example, employee records with fields such as `employee_id`, `name`, `address`, and so on.
- Use hybrid storage if your data is composed primarily of invariable XML structures, but it does contain some variable data. This means that your data contains predictably few mixed-content elements. Index the structured and unstructured parts of your data separately by using the appropriate indexes for each part (for example, an employee record that includes a free-form resume).

Choosing an XMLType Storage Model: Document-Centric



	Data-Centric	Document-Centric		
Use Case	XML schema-based data, with little variation and little structural change over time	Variable, free-form data, with some fixed embedded structures	Variable, free-form data	
Typical Data	Employee record	Technical article with author, date, and title fields	Web document or book chapter	
Storage Model	Object-Relational (Structured)	Binary XML		
Indexing	B-tree index	<ul style="list-style-type: none"> • XMLIndex with structured and unstructured components • XML Full-Text Index 	<ul style="list-style-type: none"> • XMLIndex with unstructured component index • XML Full-Text Index 	



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Document-centric

There are two use cases.

- If your data is relatively structured, your applications do not take advantage of that structure. This means that you treat the data as if it were without structure.
- Your data is, in general, without structure or of variable structure. Document structure can vary over time. *Content is mixed (many elements contain both text nodes and child elements). Many XML elements can be absent or appear in different orders. Documents may or may not conform to an XML schema.*

Use binary XML storage for all document-centric use cases.

- For general indexing of document-centric XML data, use XMLIndex indexes with unstructured components. A typical example of this use case would be an XML Web document or a book chapter.
- If your data contains some predictable, fixed structures that you query frequently, then you can use XMLIndex indexes with structured components on those parts. A typical example of this use case would be a free-form specification, with author, date, and title fields. Oracle text indexing is especially useful for document-centric cases. You learn about the different types of indexes in the lesson titled “Indexing XMLType Data.”

XMLType Storage Models: Relative Advantages

Quality	Object-Relational	Binary XML
Throughput	- XML decomposition can result in reduced throughput when ingesting or retrieving the entire content of an XML document.	+ High throughput. Fast DOM loading. There is a slight overhead from the binary XML encoder/decoder.
Queries	++ Extremely fast: relational query performance. You can create B-tree indexes on the underlying object-relational columns.	+ Streaming XPath evaluation avoids DOM construction and allows evaluation of multiple XPath expressions in a single pass. XMLIndex indexing can improve performance of XPath-based queries.
Update operations (DML)	++ Extremely fast: relational columns are updated in place.	+ In-place, piecewise update for SecureFiles LOB storage.
Space efficiency (disk)	++ Extremely space-efficient.	+ Space-efficient.
Data flexibility	- Limited flexibility. Only documents that conform to the XML schema can be stored in the XMLType table or column.	+ Flexibility in the structure of the XML documents that can be stored in an XMLType column or table.

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Object-Relational Storage Advantages

Object-relational storage advantages over the other storage models include near-relational query and update performance, optimized memory management, reduced storage requirements, B-tree indexing, and in-place updates. These advantages are at a cost of increased processing overhead during ingestion and full retrieval of XML data, and reduced flexibility in the structure of the XML that can be managed by a given XMLType table or column. Structural flexibility is reduced because data and metadata (such as column names) are separated in object-relational storage. Instance structures cannot vary easily. Object-relational storage is particularly appropriate for highly structured data whose structure does not vary, if this maps to a manageable number of database tables and joins.

XMLType Storage Models: Relative Advantages

Quality	Object-Relational	Binary XML
XML schema flexibility	<ul style="list-style-type: none"> - An XMLType table can only store documents that conform to the same XML schema. <p>In-place XML schema evolution is available, with some restrictions.</p>	<ul style="list-style-type: none"> ++ Can store XML schema-based or non-schema-based documents. An XMLType table can store documents that conform to any registered XML schemas.
Indexing support	<ul style="list-style-type: none"> + B-tree, bitmap, XML Full Text, Oracle Text, XMLIndex, and function-based indexes. 	<ul style="list-style-type: none"> XMLIndex, function-based, XML Full Text and Oracle Text indexes.
Optimized memory management	<ul style="list-style-type: none"> + XML operations can be optimized to reduce memory requirements. 	<ul style="list-style-type: none"> + XML operations can be optimized to reduce memory requirements.
Validation upon insert	<ul style="list-style-type: none"> XML data is partially validated when it is inserted. 	<ul style="list-style-type: none"> + XML schema-based data is fully validated when it is inserted.
Partitioning	<ul style="list-style-type: none"> + Available. 	<ul style="list-style-type: none"> Partition based on virtual columns.
Compression	<ul style="list-style-type: none"> ++ XML elements and attributes can be compressed individually. 	<ul style="list-style-type: none"> + XML data that uses SecureFiles LOB storage can be compressed.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Binary XML Storage Advantages

Binary XML storage provides efficient database storage, updating, indexing, and fragment extraction. Like object-relational storage, binary XML storage is aware of XML Schema data types and can take advantage of native database data types. Like object-relational storage, binary XML storage allows for piecewise updates. Unlike object-relational storage, because binary XML data can also be used outside the database it can serve as an efficient XML exchange medium, and you can off-load work from the database to increase overall performance in many cases. Unlike object-relational storage, binary XML data is kept in document order. Like object-relational storage, data and metadata can be separated at the database level, for efficiency. However, binary XML storage allows for intermixed data and metadata, which lets instance structures vary. Binary XML storage allows for very complex and variable data, which in the object-relational storage model could necessitate using many database tables. Unlike object-relational storage, you can use binary XML storage for XML schema-based data even if the XML schema is not known beforehand, and you can store multiple XML schemas in the same table and query across common elements.

Lesson Agenda

- Using XMLType
- Choosing an XMLType storage model
 - Binary XML storage
 - Object-relational storage
- Loading data into XMLType
- Specifying SQL constraints



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Loading Data into XMLType

You can load data into XMLType columns by using:

- An INSERT with the XMLType () constructors by using a VARCHAR2, CLOB, BLOB, or a BFILE:

```
INSERT INTO emp_resumes VALUES (100,
    XMLType(
        '<?xml version="1.0"?>
        <RESUME>
            <FULL_NAME>Steven King</FULL_NAME>
            <JOB_HISTORY>
                <JOB_ID>AD_PRES</JOB_ID>
            </JOB_HISTORY>
        </RESUME>' ) ;
```

1 rows inserted

- Utilities such as:
 - Oracle XML/SQL Utility for Java, FTP, and WebDAV
 - Oracle export, import, and SQL*Loader

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use a simple INSERT operation in SQL or PL/SQL to load an XML document into the database. Before the document can be stored as an XMLType column or table, it must be converted into an XMLType instance by using one of the XMLType constructors.

Loading data into an XMLType column by using the INSERT statement, as shown in the slide, creates a row containing the resume XML document for employee Steven King by using a VARCHAR2 argument in the XMLType () constructor. Constructors replace the XMLType.createXML () function for inserting XML data into the column, as in the following example:

```
INSERT into emp_resumes
VALUES (101, XMLType.createXML('<?xml version="1.0"?>
<RESUME>
    <FULL_NAME>Neena Kochhar</FULL_NAME>
    <JOB_HISTORY><JOB_ID>AD_VP</JOB_ID></JOB_HISTORY>
</RESUME>' )) ;
```

1 rows inserted

Loading Data into XMLType

- You can also load the XML data into XMLType columns by using a BFILE, as follows:

```
INSERT INTO emp_resumes VALUES (102,
    XMLType(BFILENAME ('XML_DIR','code_04_31_s.xml'),
    NLS_CHARSET_ID ('AL32UTF8')));
```

```
1 rows inserted  
committed
```

- Before inserting the XML document as shown in the preceding example, you must create a SQL directory object that points to the directory containing the file to be processed (already created in this class).

```
CREATE DIRECTORY xml_dir AS '/home/oracle/labs/xml_dir';
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Before inserting the XML document as shown in the slide example, you must create a SQL directory object that points to the directory containing the file to be processed. In this course, we created a directory object named `xml_dir` as follows:

```
CREATE DIRECTORY xml_dir AS '/home/oracle/labs/xml_dir';
```

Oracle Database allows the creation of XML objects as resources in Oracle XML DB by using the WebDAV and FTP utilities to load and access the documents in the database. Oracle XML/SQL Utility for Java offers a command-line interface for loading the XML data.

The `code_04_31_s.xml` file contains the following information:

```
-<RESUME>
  <FULL_NAME>Lex De Haan</FULL_NAME>
  -<JOB_HISTORY>
    <JOB_ID>AD_VP</JOB_ID>
  </JOB_HISTORY>
-</RESUME>
```

For more information about loading XML content into Oracle XML DB, see the *Oracle XML DB Developer's Guide 12c Release 1 (12.1)*.

Specifying SQL Constraints with O-R and Binary XML Storages

```
-- Execute in oe/oe
DECLARE
  depositschema VARCHAR2(2000) :=
'<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xdb="http://xmlns.oracle.com/xdb">
<xss:element name="Deposit">
  <xss:complexType xdb:SQLType="Acct">
    <xss:sequence>
      <xss:element name="Dep_No" type="xs:float" nullable="false"/>
      <xss:element name="Dep_Id" type="xs:float"/> // Line 1
      <xss:element name="Bal" type="xs:float" default="0" nullable="false"/>
      <xss:element name="Limit" type="xs:float"/>
      <xss:element name="Dep_Date" type="xs:date"/>
      <xss:element name="Dep_Type" type="xs:string"/>
      <xss:element name="Dep_Open" type="xs:boolean"/>
    </xss:sequence>
  </xss:complexType>
</xss:element>
</xs:schema>';
BEGIN
  DBMS_XMLSHEMA.RegisterSchema('deposit.xsd', depositschema );
END;
/
```

deposit.xsd
registered
schema

1

```
CREATE TABLE deposits OF XMLType( CONSTRAINT "Dep_Id_PK"
PRIMARY KEY (xmldata."Dep_Id") XMLSCHEMA
"deposit.xsd" ELEMENT "Deposit",
```

2

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Specifying SQL Constraints

You can specify SQL constraints on XMLType tables and columns when O-R storage or binary XML storage is selected. You can specify constraints either at the time of table creation or after table creation. You might want to add constraints to an XMLType table that is based on an XML schema.

Example 1 in the slide shows the deposit.xsd registered schema. The output of this code examples is as follows:

anonymous block completed

Example 2 shows how to create the deposits table with the primary key defined on Dep_Id. The output of this code examples is as follows:

table DEPOSITS created.

Defining Constraints on Binary XML Data

Perform the following steps:

1. Define virtual columns for the XML data that you are interested in.
2. Use virtual columns to constrain the XMLType data as a whole.

You can create a virtual column based on an XML element or attribute by defining it in terms of a SQL expression that involves that element or attribute.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To constrain XML data according to the values of individual elements or attributes, you cannot use the standard approach that you use for relational data. This is because individual XML elements and attributes are not mapped to individual database columns or tables. But you can constrain XML data by using a different technique (as explained in the two steps in the slide). This approach applies only to XML data that is stored as binary XML.

You can create a virtual column on XMLType data as you would create using any other type of data, but by using a slightly different syntax. However, you cannot specify any constraints in association with the column definition. You can create a virtual column based on an XML element or attribute by defining it in terms of a SQL expression that involves that element or attribute. This means that you create a function-based column.

Note: The virtual columns that you create on an XMLType table are hidden and do not show up when you use the DESCRIBE statement. If you create virtual columns on a table that has an XMLType column, you see the virtual columns along with the non-virtual columns listed by a DESCRIBE operation.

Defining Constraints on Binary XML Data

```

CREATE TABLE part_binaryxml OF XMLType
XMLTYPE STORE AS BINARY XML
VIRTUAL COLUMNS
(c_reference AS
  (XMLCast(XMLQuery('/PurchaseOrder/Reference' PASSING
    OBJECT_VALUE RETURNING CONTENT)AS VARCHAR2(30)))
/
INSERT INTO part_binaryxml
SELECT OBJECT_VALUE FROM OE.purchaseorder;

```

table PART_BINARYXML created.
132 rows inserted.

```

ALTER TABLE part_binaryxml ADD CONSTRAINT
reference_is_unique UNIQUE (c_reference);

```

ALTER TABLE part_binaryxml succeeded.

```

INSERT INTO part_binaryxml
VALUES (XMLType(bfilename('XML_DIR', 'DemoReference.xml'),
nls_charset_id('AL32UTF8')));

```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The examples in the slide show how to define a unique constraint on an XMLType table that is stored as binary XML, how to insert sample data, and how to display a constraint violation error. The first step shows how to create an XMLType table part_binaryxml and define a virtual column by using the Reference element in a purchase order document. The example also illustrates populating the purchaseorder table data into the part_binaryxml table.

The second step shows how to alter the table definition by adding the unique constraint. The unique constraint reference_is_unique ensures that the value of the /PurchaseOrder/Reference/text() node is unique across all the documents stored in the part_binaryxml table. If you insert data from the DemoReference.xml file into the part_binaryxml table by using the INSERT statement as shown in the third step, you see the following constraint violation error:

```

INSERT INTO part_binaryxml
*
ERROR at line 1:
ORA-00001: unique constraint (OE.REFERENCE_IS_UNIQUE) violated

```

You learn how to use XMLCast in the lesson “Using XQuery to Retrieve XML Data in Oracle XML DB.”

Quiz

You can use XMLType to:

- a. Query XML data
- b. Provide efficient XPath access
- c. Shield applications from storage models
- d. Prepare for future optimizations
- e. Provide structured storage with DOM
- f. Store only XML content



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a, b, c, d, e

Summary

In this lesson, you should have learned how to:

- Describe XMLType
- Create XMLType objects
- Describe the XML schema support in Oracle XML DB
- Describe XMLType storage options
- Load data into XMLType
- Specify SQL constraints on XMLType tables



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In this lesson, you learned how the XMLType data type is an integral part of the Oracle XML database environment. You learned to create XMLType columns, object tables of XMLType instances, and object types with an XMLType attribute. You saw the different storage models available in Oracle XML DB. You learned how to register an XML schema and how to delete a registered XML schema in Oracle XML DB. You also learned how to load data into XMLType tables. You learned the three types of storage models: structured, unstructured, and binary XML. You also learned how to specify SQL constraints on XMLType tables.

Practice 4: Overview

This practice covers the following topics:

- Loading data into XMLType tables
- Specifying SQL constraints



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Using XML Schema with Oracle XML DB



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the XML schema support in Oracle XML DB
- Create XML schema-based XMLType tables
- Register an XML schema
- Delete a registered XML schema
- Use XML schema evolution to manage changes in an XML schema
- Perform copy-based and in-place XML schema evolution



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Using XML schema with Oracle XML DB
- Creating XML schema-based XMLType tables
- Registering an XML schema in Oracle XML DB
- Deleting a registered XML schema
- Using XML schema evolution to manage changes in an XML schema
- Performing copy-based and in-place XML schema evolution



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

W3C XML Schema Recommendation: Overview

- An XML schema can be considered as the metadata that describes a class of XML documents.
- The term instance document is often used to describe an XML document that conforms to a given XML schema.
- The W3C Schema working group publishes an XML schema, often referred to as the "Schema for Schemas."
 - This XML Schema provides the definition, or vocabulary, of the XML Schema language.
 - All valid XML schemas can be considered to be members of the class defined by this XML Schema.
 - An XML schema is therefore an XML document that conforms to the class defined by the XML Schema published at: <http://www.w3.org/2001/XMLSchema>.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The W3C XML Schema Recommendation defines a standardized language for specifying the structure, content, and certain semantics of a set of XML documents. An XML schema can be considered the metadata that describes a class of XML documents. The XML Schema Recommendation is described at: <http://www.w3.org/TR/xmlschema-0/>

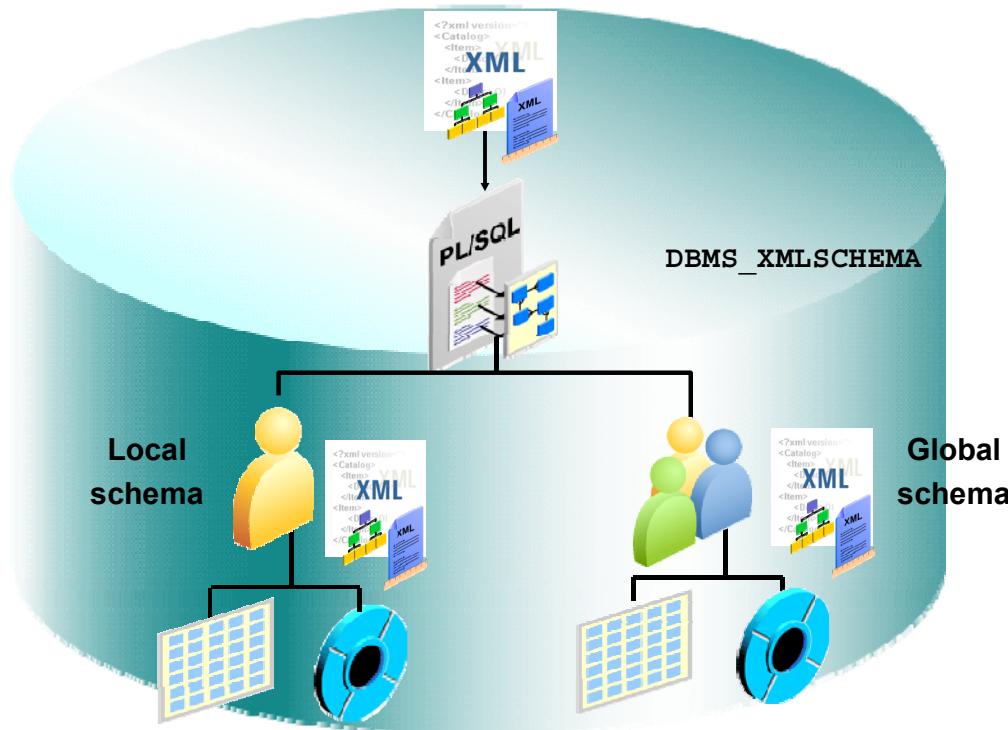
XML Instance Documents

Documents conforming to a given XML schema can be considered as members or instances of the class defined by that XML schema. Consequently, the term instance document is often used to describe an XML document that conforms to a given XML schema. The most common use of an XML schema is to validate that a given instance document conforms to the rules defined by the XML schema.

XML Schema for Schemas

The W3C Schema working group publishes an XML Schema, often referred to as the "Schema for Schemas". This XML Schema provides the definition, or vocabulary, of the XML Schema language. All valid XML schemas can be considered to be members of the class defined by this XML Schema. An XML schema is thus an XML document that conforms to the class defined by the XML schema published at:
<http://www.w3.org/2001/XMLSchema>.

XML Schema Support in Oracle Database 12c



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle Database provides XML schema support. You can perform the following tasks:

- Register W3C-compliant local and global XML schema.
- Validate your XML documents against registered XML schema definitions.
- Generate XML schema from object types.
- Reference an XML schema owned by another user.
- Explicitly reference a global XML schema when a local XML schema exists with the same name.
- Generate a structured database mapping from your XML schemas during XML schema registration. This includes generating SQL object types, collection types, and default tables, and capturing the mapping information by using XML schema attributes.
- Specify a particular SQL type mapping when there are multiple legal mappings.
- Create XMLType tables, views, and columns based on registered XML schemas.
- Perform manipulation (data manipulation language) and queries on XML schema-based XMLType tables.
- Automatically insert data into default tables when schema-based XML instances are inserted into the Oracle Database Repository by using FTP, HTTP, or SQL.

XML Schema and Oracle XML DB

- Creating XML schema-based tables and columns
- Creating constraints on XML tables and columns
- Using queries and DML on XML tables and columns
- Using Oracle XML schema annotations
- Generating SQL types
 - A SQL object type is generated for each `ComplexType` defined in the XML schema
 - The definition of the SQL object mirrors the definition of `ComplexType`
 - Each child element and attribute defined by `ComplexType` is mapped to an attribute of the SQL object type



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Creating XML Schema-Based Tables and Columns

You can create `XMLType` tables based on an XML schema that is already registered. Hidden columns are created corresponding to the object type to which the primary key element is mapped. In addition, an `XMLExtra` object column is created to store top-level instance data, such as namespaces declarations and so on.

Creating Constraints on XML Tables and Columns

You can create unique and referential integrity constraints on XML schema-based `XMLType` tables. The constraints are enforced every time a change is made to the stored XML documents.

Querying and DML on XML Tables and Columns

New instances can be inserted into an `XMLType` table. The `XMLType` table can be queried by using the XPath-based SQL operators. The XPath rewrite mechanism rewrites queries that involve XPath arguments to SQL operators to access the underlying object attribute columns directly, thereby avoiding construction of XML followed by subsequent XPath evaluation.

Using Oracle Schema Annotations

Using Oracle XML DB, application developers and database administrators can annotate the XML schema because they are given greater control over the set of objects that is generated from the XML schema. Annotations are covered in more details in the lesson titled “Oracle XML DB Manageability.”

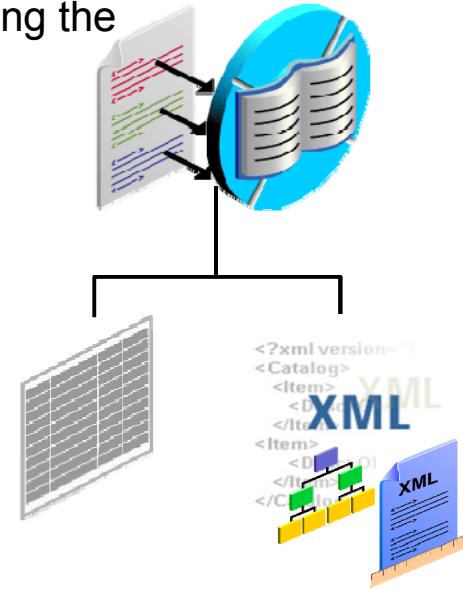
Generating SQL Types

During the schema registration process:

- Oracle XML DB generates a SQL object type for each `ComplexType` defined in the XML schema
- The definition of the SQL object mirrors the definition of `ComplexType`
- Each child element and attribute defined by `ComplexType` is mapped to an attribute of the SQL object type

XMLType and XML Schema

- When creating an `XMLType` instance, you can specify that it conforms to a preregistered XML schema.
- An XML schema is registered by using the `DBMS_XMLSHEMA` package.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In Oracle Database 12c, you can create `XMLType` tables and columns. For example, you can (optionally) specify that they:

- Conform to a preregistered XML schema
- Are stored in the object-relational format specified by the XML schema

You can also choose to wrap existing relational and object-relational data into XML format by using `XMLType` views.

You can store an `XMLType` object as:

- XML schema-based objects, which can be stored in object-relational format in tables, columns, and views
- Non-XML-schema-based objects, which are stored as binary XML

Mapping from an XML instance document to object-relational or binary XML storage is defined by registering your XML schema in Oracle Database 12c by using the `DBMS_XMLSHEMA` package. You must register the XML schema before you store XML schema-based instance documents. To reference a registered XML schema, use its URL. You can also store an `XMLType` object in binary xml format.

Oracle XML DB supports the use of the W3C XML schema in two ways:

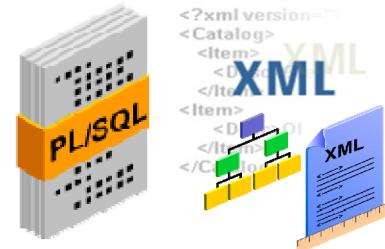
- Automatic validation of instance documents
- Definition of storage models

To use a W3C XML schema with Oracle XML DB, the XML schema document must be registered with the database. After you register an XML schema, you can create XMLType tables and columns that conform to the schema.

XML schemas are registered by using methods provided by the `DBMS_XMLSHEMA` PL/SQL package. XML schemas can be registered as global or local schemas. This is covered later in this lesson.

XML Schema Management

- The XML schema functionality is available through DBMS_XMLSHEMA.
- You can use the DBMS_XMLSHEMA subprograms to do the following:
 - Register an XML schema
 - Delete a previously registered XML schema
 - Recompile a previously registered XML schema
 - Generate an XML schema
 - Evolve an XML schema



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XML schema functionality of Oracle Database12c is available through the PL/SQL-supplied package, DBMS_XMLSHEMA, which is a server-side component that handles the registration of XML schema definitions for use by Oracle Database 12c applications.

You can use the REGISTERSCHEMA procedure to register an XML schema that is specified as a VARCHAR2 or XMLType. The REGISTERSCHEMA procedure registers an XML schema by associating an XML schema document with a URL.

You can re-compile an already registered XML schema. This is useful for bringing a schema in an invalid state to a valid state by using the COMPILESCHEMA procedure.

You can generate an XML schema from an object type by using the GENERATESCHEMA or GENERATESCHEMAS procedures.

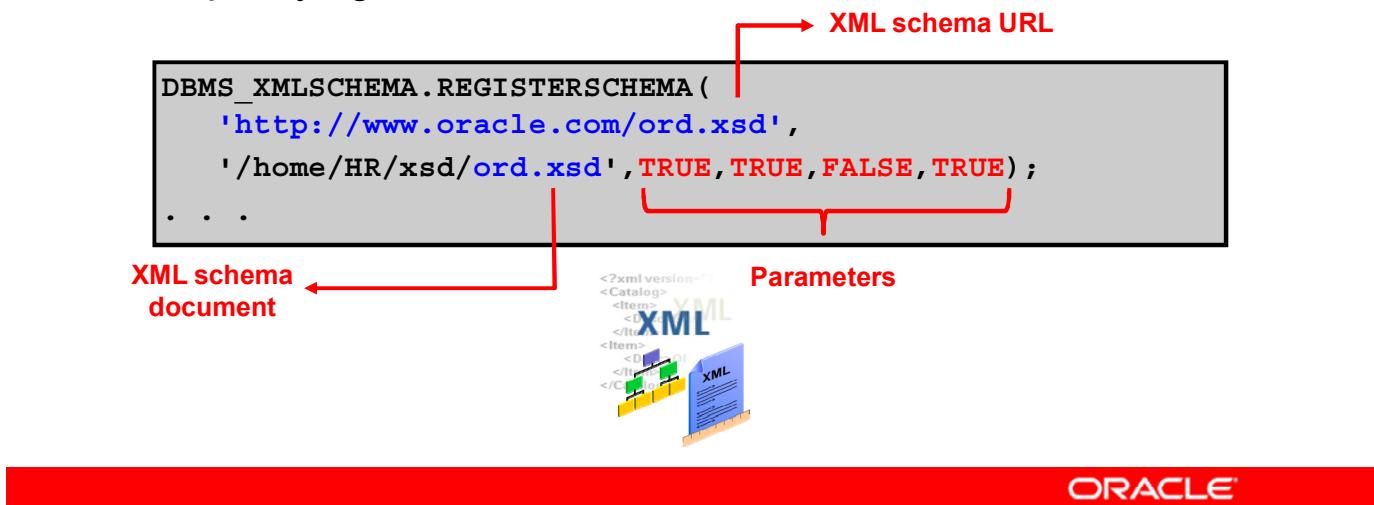
You delete a registered XML schema by using the DELETESCHEMA procedure. When you attempt to delete an XML schema, DBMS_XMLSHEMA checks whether the current user has the appropriate privileges and whether there are any dependents.

A major challenge when using an XML schema is how to deal with the changes in the content or structure of XML documents. For example, new elements or attributes may need to be added to an XML schema definition. In such cases, you perform XML schema evolution so that new requirements are accommodated, while existing instance documents remain valid and existing applications can continue to run.

Registering an XML Schema

The REGISTERSCHEMA procedure registers an XML schema for use by the database. You provide the following two items:

- The XML schema document (.xsd)
- The URL that will be used by the XML documents specifying conformance with the XML schema



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To register an XML schema, you must provide two items:

- The XML schema document
- A string that can be used as a unique identifier for the XML schema, after it is registered with Oracle Database. Instance documents use this unique identifier to identify themselves as members of the class defined by the XML schema. The identifier is typically in the form of a URL, and is often referred to as the **schema location hint** or **document location hint**. The URL is provided in the root element of the instance document by using one of the following:
 - noNamespaceSchemaLocation attribute
 - schemaLocation attribute as defined in the W3C XML schema recommendation

XML schemas are registered by using methods provided by the DBMS_XMLSHEMA PL/SQL package. XML schemas can be registered as global or local schemas.

The slide example registers the XML schema with the URL specified in the first parameter, that is, `http://www.oracle.com/ord.xsd`.

Before executing the example in the slide, the XML schema document `ord.xsd` is first loaded either by FTP or WebDav into the database. The XML schema document is loaded to the location `/home/HR/xsd/ord.xsd`. You learned how to use WebDav in the *Oracle 12c: XML Fundamentals* prerequisite course.

Registering an XML Schema Example: Method 1

```
BEGIN  
  DBMS_XMLSHEMA.REGISTERSCHEMA(  
    'http://www.oracle.com/emp.xsd',  
    '<xs:schema version="1.0"  
      xmlns:xs="http://www.w3.org/2001/XMLSchema"  
      xmlns:xdb="http://xmlns.oracle.com/xdb">  
      <xs:complexType name="ActionsType"  
        xdb:SQLType="XDBPO_ACTIONS_TYPE">  
        <xs:sequence>  
          <xs:element name="Action" maxOccurs="4"  
            xdb:SQLName="ACTION"  
            . . .  
          </xs:element>  
        </xs:schema>');  
  . . .  
END;  
/
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The partial slide example syntax shows how you can use the XML schema document as the second parameter.

Registering an XML Schema Example: Method 2

```
BEGIN  
  DBMS_XMLSHEMA.REGISTERSCHEMA(  
    SCHEMABURL =>'http://www.oracle.com/emp.xsd',  
    SCHEMADOC => bfilename('XML_DIR','emp.xsd'),  
    CSID => nls_charset_id('AL32UTF8'));  
END;  
/
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The partial slide example syntax shows how you can register an XML schema by using the BFILE mechanism to read the source document from a file on the local file system of the database server.

The REGISTERSCHEMA Procedure Parameters

Parameter	Description
schemaURL	URL that uniquely identifies the schema document. This value is used to derive the path name of the schema document within the XDB hierarchy.
schemaDoc	A valid XML schema document
local	Specifies whether the schema is local (default) or global. A local schema is added in the <code>/sys/schemas/<username></code> folder. A global schema is added in the <code>/sys/schemas/PUBLIC</code> folder.
genTypes	Specifies whether the schema compiler must generate object types (TRUE).
genTables	Specifies whether the schema compiler must generate default tables (TRUE).
force	If this parameter is set to TRUE , the XML schema registration does not raise errors. Instead, it creates an invalid XML schema object in case of any errors (FALSE).
owner	Specifies the name of the database user who owns the XML schema object. By default, the user registering the XML schema owns the XML schema object.
Options	Provides additional options to specify how the schema should be registered. The <code>REGISTER_BINARYXML</code> option registers the schema for binary XML.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

schemaURL: (IN) URL that uniquely identifies the schema document. This value is used to derive the path name of the schema document within the XDB hierarchy.

schemaDoc: (IN) A valid XML schema document

local: (IN) Specifies whether the schema is local (default) or global. A local schema is added in `/sys/schemas/<username>`. If a schema is registered as global, it is added in the `/sys/schemas/PUBLIC` folder for which the user needs write privileges.

genTypes: (IN) Specifies whether the schema compiler must generate object types. This is significant only when you register an XML schema for data that is stored o-r.

genTables: (IN) Specifies whether the schema compiler must generate default tables. `genTypes` is significant only when you register an XML schema for data that is stored o-r.

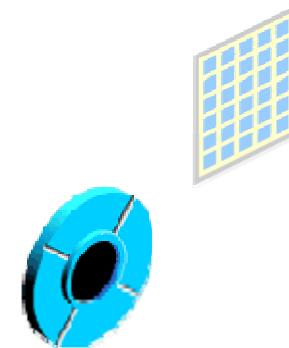
force: (IN) If this parameter is set to **TRUE**, the XML schema registration will not raise errors. Instead, it creates an invalid XML schema object in case of any errors.

owner: (IN) Specifies the name of the database user who owns the XML schema object. By default, the user registering the XML schema owns the XML schema object. This parameter can be used to register an XML schema to be owned by a different database user.

options: Provides additional options to specify how the schema should be registered. The `REGISTER_BINARYXML` option registers the schema for Binary XML.

Storage and Access Infrastructure

- When registering an XML schema, Oracle XML DB:
 - Creates types
 - Creates default tables
- This is controlled by arguments to the REGISTERSCHEMA procedure.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

While registering an XML schema, Oracle Database 12c also performs the following steps to facilitate storing, accessing, and manipulating XML instances that conform to the XML schema:

- **Creating Types:** When an XML schema is registered, the appropriate SQL object types are created to enable the o-r storage of XML documents that conform to the XML schema. Use the Oracle Database 12c-defined attributes in the XML schema documents to control how these object types are generated. For binary XML, the XML schema types are mapped to the binary XML encoding types.
- **Creating Default Tables:** As part of the XML schema registration, Oracle Database 12c generates default XMLType tables for all root elements. You can also specify any column- and table-level constraints for use during table creation.

You can control the creation of database objects and suppress their generation by arguments to the registerSchema procedure. For example: Use xdb:SQLType to specify the name of the SQL type to be generated. Set xdb:defaultTable to the empty string ("") to suppress the creation of the default table for a particular root element.

Local and Global XML Schemas

- An XML schema can be registered as:
 - Local: Visible only to the owner (default)
 - Global: Visible and usable by all database users
- To register a global XML schema, set the LOCAL parameter value to FALSE.

```
BEGIN
  DBMS_XMLSHEMA.registerSchema(
    SCHEMABURL =>
      'http://xmlns.oracle.com/xdb/global_po.xsd',
    SCHEMADOC => bfilename('XML_DIR',
      'purchaseOrder.xsd'),
    LOCAL => FALSE,
    GENTYPES => TRUE,
    genTables => FALSE,
    force => FALSE
  );
END;
```



local



global

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- **Local XML schema:** The registered XML schema is visible only to the owner.
- **Global XML schema:** The registered schema is visible and usable by all database users. To register a global schema, a user must be granted the XDB_ADMIN role.

When you register an XML schema, DBMS_XMLSHEMA adds an Oracle database resource corresponding to the XML schema into the Oracle XML DB Repository. The XML schema URL determines the path name of the resource in the repository.

The slide example demonstrates how to register an XML schema as a global schema by setting the local parameter to FALSE in the REGISTERSCHEMA procedure.

The output of the slide example is as follows:

```
anonymous block completed
Connection created by CONNECT script command disconnected
```

Deleting an XML Schema

- Use the following syntax of the DBMS_XMLSHEMA.DELETESCHEMA procedure:

```
EXEC DBMS_XMLSHEMA.DELETESCHEMA (
    'http://www.oracle.com/ord.xsd',
    DBMS_XMLSHEMA.DELETE CASCADE FORCE);
```

XML schema URL

Delete option

- This procedure does the following:
 - Checks for appropriate current user privileges
 - Checks for dependent tables
 - Removes the XML schema document from the /sys/schemas folder from Oracle XML DB Repository
 - Removes the XML schema document from DBA_XML_SCHEMAS
 - Drops the default table or raises an error



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A registered XML schema can be deleted by using the DBMS_XMLSHEMA.DELETESCHEMA procedure. When you attempt to delete an XML schema, use DBMS_XMLSHEMA.DELETESCHEMA, which does the following:

1. Checks whether the current user has the appropriate privileges to delete the resource corresponding to the XML schema within Oracle XML DB Repository
2. Checks whether there are any tables dependent on the XML schema that is to be deleted
3. Removes the XML schema document from the /sys/schemas folder in the Oracle XML DB Repository
4. Removes the XML schema document from DBA_XML_SCHEMAS, unless it was registered for use with binary XML instances and neither delete_invalidate nor delete_cascade_force is used. The
5. Drops the default table or raises an error

Parameters for the DELETESCHEMA Procedure

Parameter	Description
<code>schemaURL</code>	A URL identifying the schema to be deleted
<code>delete_option</code>	An option for deleting the schema as follows: <ul style="list-style-type: none"> DELETE_RESTRICT: Schema deletion fails if there are any tables or schemas that depend on this schema. DELETE_INVALIDATE: Schema deletion does not fail if there are dependencies. Instead, it simply invalidates all the dependent objects. DELETE CASCADE: Schema deletion also drops all the dependent SQL types and tables. However, deletion fails if there is any data in the tables that conforms to this schema. DELETE CASCADE_FORCE: This is similar to CASCADE, except that it does not check for any stored data conforming to this schema. Also, it ignores errors.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Purging an XML Schema

- You must call `DBMS_XMLSHEMA.PURGESCHEMA()` to remove the XML schema completely from the system. The syntax of the PURGESCHEMA procedure is as follows:
`DBMS_XMLSHEMA.PURGESCHEMA (Schema_id IN RAW);`
- You can obtain `SCHEMA_ID` from the `USER_XML_SCHEMAS` view.
`DBMS_XMLSHEMA.purgeSchema()` removes the XML schema completely from `DBA_XML_SCHEMAS`. Before you use `DBMS_XMLSHEMA.PURGESCHEMA`, make sure that you transform all the existing XML documents that reference the XML schema to be purged. This means that you make them reference a different XML schema or no XML schema.
- Note:** It is recommended that, in general, you use `DELETE_RESTRICT` or `DELETE CASCADE`. When you are sure you no longer need the XML schema, call `DBMS_XMLSHEMA.PURGESCHEMA` instead of `DELETE CASCADE_FORCE`.

For more information about this topic, refer to the following documentation references:

- Oracle Database PL/SQL Packages and Types Reference 12c Release 1 (12.1)*
- Oracle 12c XML API Reference—PL/SQL*
- Oracle 12c XML Database Developer's Guide*

Registering an XML Schema for Binary XML

```
-- To register an XML schema for binary XML:  
-- Specify the DBMS_XMLSCHEMA.REGISTER_BINARYXML option  
-- and set the GENTYPES parameter to FALSE.  
BEGIN  
  DBMS_XMLSCHEMA.registerSchema(  
    SCHEMABURL =>  
      'http://xmlns.oracle.com/xsd/po_bin.xsd',  
    SCHEMADOC =>  
      bfilename('XML_DIR','po_bin.xsd'),  
    CSID => nls_charset_id('AL32UTF8'),  
    local => TRUE,  
    genTypes => FALSE,  
    genTables => FALSE,  
    force => FALSE,  
    OPTIONS => DBMS_XMLSCHEMA.REGISTER_BINARYXML  
  );  
END;  
/
```

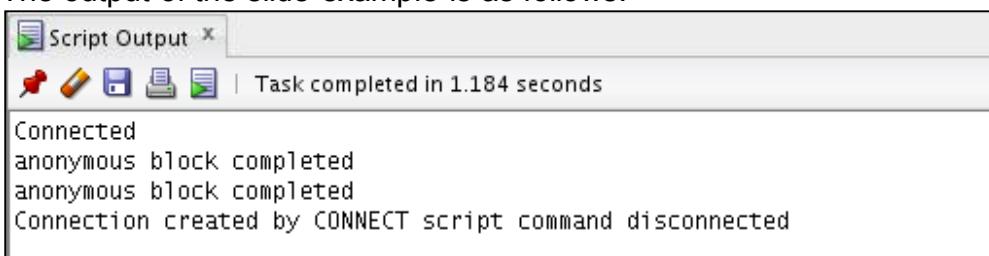


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example registers the XML schema with the URL specified as `http://xmlns.oracle.com/xsd/po_bin.xsd`. You can use FTP to load the `po_bin.xsd` schema definition document into the database before executing the PL/SQL procedure. The most important option is `DBMS_XMLSCHEMA.REGISTER_BINARYXML`, which indicates that the XML schema is used for binary XML storage.

If you register an XML schema by using the `REGISTER_BINARYXML` option, the XML instance documents that reference that XML schema are stored as binary XML and their XML schema data types are encoded in the binary XML data. The `genTables` parameter must be set to `False`.

Different XML schemas specify different binary XML encodings. When you use the `ALLOW ANYSCHEMA` option, XML documents that reference different XML schemas—and that might therefore have different binary XML encodings—can be stored in the same `XMLType` column or table. You learn how to use the `ALLOW ANYSCHEMA` option later in this lesson. The output of the slide example is as follows:



The screenshot shows the Oracle SQL Developer interface with a "Script Output" window. The window title is "Script Output". Below the title bar, there are icons for a script file, a red X, a pencil, a folder, and a refresh symbol. A status message "Task completed in 1.184 seconds" is displayed. The main pane of the window shows the following text:
Connected
anonymous block completed
anonymous block completed
Connection created by CONNECT script command disconnected

DBMS_XMLSHEMA.purgeSchema()

If you have an XML schema that is registered for use with binary XML:

- The XML schema is not removed from DBA_XMLSchemas when you delete it by using the `delete_restrict` or `delete_cascade` options.
- Using the above options, the XML schema document continues to exist in the XML DB dictionary.
- You cannot use the XML schema to encode any new XML documents.
- XML DB can decode any old XML documents that reference the deleted XML schema.
- You must call `DBMS_XMLSHEMA.purgeSchema()` to remove the XML schema completely from the system.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If you have an XML schema that is registered for use with binary XML, it is not removed from DBA_XMLSchemas when you delete it by using the `delete_restrict` or `delete_cascade` options. When you use the `delete_restrict` or `delete_cascade` options, the XML schema document continues to exist in the XML DB dictionary. You cannot use the XML schema to encode any new XML documents. But the system would be able to decode any old XML documents that reference the deleted XML schema. You must call `DBMS_XMLSHEMA.purgeSchema()` to remove the XML schema completely from the system.

Creating XML Schema–Based XMLType Tables

- You can create XMLType tables and columns that conform to an XML schema registered with Oracle XML DB.
- XML schema–based XMLType tables and columns provide advantages such as the following:
 - Only XML documents that validate against the XML schema are stored in the database.
 - They enable optimized query and update processing of XML.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

With Oracle XML DB, you can create XMLType tables and columns that are constrained to a global element defined by a registered XML schema. This has several advantages, such as the following:

- Oracle XML DB ensures that only those XML documents that validate against the XML schema are stored in the column or table.
- The contents of the table or column conform to a known XML structure. Therefore, Oracle XML DB uses this information contained in the XML schema to provide optimized query and update processing of XML.
- The object-relational model used to store the document is derived from the XML schema.

After an XMLType column is constrained to a particular element and a particular XML schema, it can contain only those documents that are compliant with the schema definition of that element.

Creating XML Schema-Based XMLType Tables: Example

```
BEGIN
  DBMS_XMLSHEMA.registerSchema(SCHEMABURL =>
    'http://xmlns.oracle.com/code_ex/po_test.xsd',
    SCHEMADOC => bfilename('XML_DIR',
      'purchaseOrder.xsd'),
    LOCAL => TRUE,
    GENTYPES => TRUE,
    genTables => FALSE,
    force => FALSE
  );
END;
/
CREATE TABLE po_code_ex OF XMLType
  XMLSCHEMA "http://xmlns.oracle.com/code_ex/po_test.xsd"
  ELEMENT "PurchaseOrder";
```

Connected
anonymous block completed

*Action:
table PO_CODE_EX created.
Connection created by CINNEMT script command disconnected

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can constrain an XMLType table column to a particular element and XML schema by adding the appropriate XMLSCHEMA and ELEMENT clauses to the CREATE TABLE operation.

The example in the slide shows a CREATE TABLE statement that creates the po_code_ex XMLType table. In the example, the XMLType instance is constrained to the PurchaseOrder element that is defined by the XML schema registered with the URL http://xmlns.oracle.com/code_ex/po_test.xsd.

The following example creates the po_code_ex_reltabLe relational table that has the xml_document XMLType column.

```
CREATE TABLE po_code_ex_reltabLe (id NUMBER, xml_document XMLType)
  XMLTYPE COLUMN xml_document
  XMLSCHEMA "http://xmlns.oracle.com/code_ex/po_test.xsd"
  ELEMENT "PurchaseOrder";
```

In the po_code_ex_reltabLe table, the XMLType instance is constrained to the PurchaseOrder element that is defined by the XML schema that is registered with the URL http://xmlns.oracle.com/code_ex/po_test.xsd.

The data associated with an XMLType table or column that is constrained to an XML schema can be stored in two different ways:

- Decomposed and stored object-relationally (structured storage)
- Stored as binary XML by using a single binary XML column (binary XML storage)

When you create a table that stores XML instance documents that reference an XML schema, you can specify the storage options to use. The default storage model is the structured storage model. To override this default model, and use the binary XML storage instead, you can use the STORE AS clause in the CREATE TABLE statement.

Note: When registering an XML schema, if you specify the option as REGISTER_BINARYXML, no other storage model is allowed in the STORE AS clause. You learn how to specify binary XML storage on the next few pages.

Quiz

Which of the following statements are true:

- a. To use a W3C XML schema with Oracle XML DB, the XML schema document must be registered with the database.
- b. After you register an XML schema, you can create XMLTYPE tables and columns that conform to the schema.
- c. To register an XML schema, you must provide the XML schema document and the URL that will be used by the XML documents specifying conformance with the XML schema.
- d. XML schemas are registered by using the DBMS_XMLSHEMA PL/SQL package.
- e. XML schemas can be registered as global schemas only.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a, b, c, d

Practice 5-1: Overview

This practice covers the following topics:

- Registering an XML schema in Oracle XML DB
- Deleting an XML schema that is registered in Oracle XML DB



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 5-2: Overview

This practice covers the following topics:

- Creating XML schema-based XMLType tables and columns
- Loading data into the newly created XMLType tables



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Using XML schema with Oracle XML DB
- Creating XML schema-based XMLType tables
- Registering an XML schema in Oracle XML DB
- Deleting a registered XML schema
- Using XML schema evolution to manage changes in an XML schema
- Performing copy-based and in-place XML schema evolution



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XML Schema Evolution

- The XML schema evolution:
 - Can be used to manage changes to an XML schema
 - Is useful when the structure or format of XML documents must be changed to reflect the changes in application requirements
- Oracle XML DB supports two kinds of XML schema evolution:
 - Copy-based XML schema evolution
 - In-place XML schema evolution



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When using XML schema with Oracle XML DB, you (as an application developer) face the challenge of dealing with the issues related to changes in the content or structure of the XML documents. For example, you can make changes such as the following:

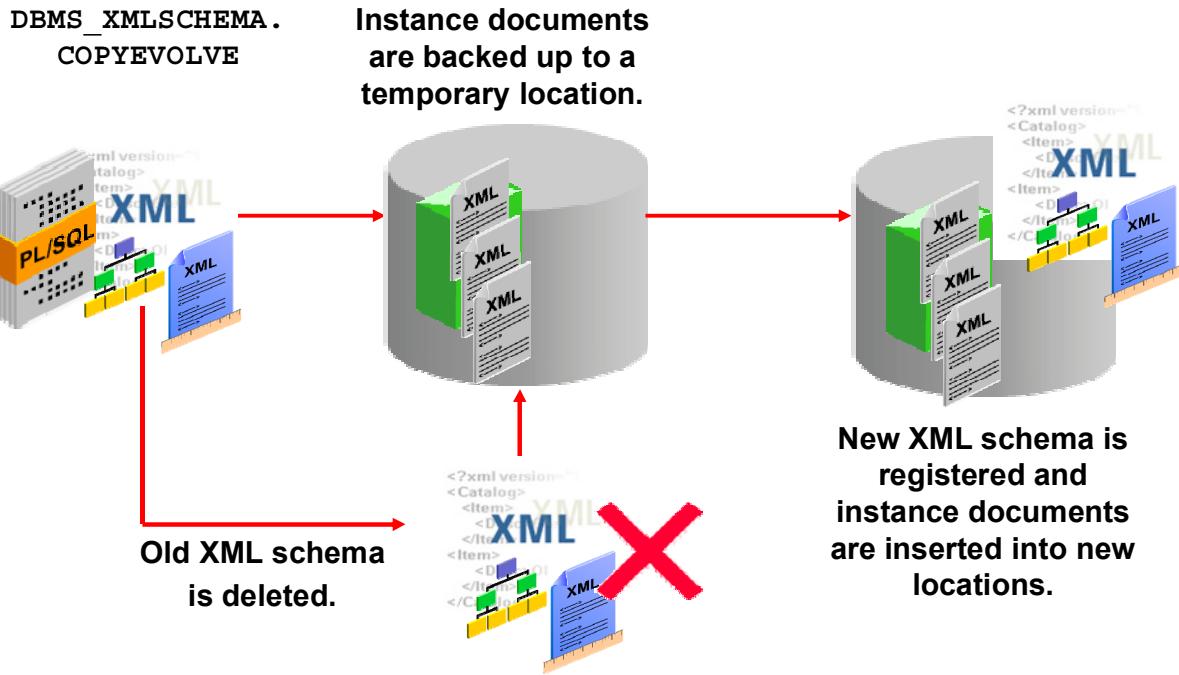
- Add new elements or attributes to an XML schema definition
- Modify a data type
- Relax or tighten certain minimum and maximum occurrence requirements

You need to manage changes in an XML schema so that new requirements are accommodated, while any existing instance documents remain valid (or can be made valid automatically), and existing applications can continue to run.

If you do not care about the existing documents, you can drop the `XMLType` tables that are dependent on the old XML schema, delete the old XML schema, and register the new XML schema at the same URL.

Oracle XML DB provides an *XML schema evolution* technique that can be used to manage the changes to an XML schema. This is a very useful technique because it provides flexibility in designing and managing changes in XML schemas.

Copy-Based XML Schema Evolution



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

With the copy-based XML schema evolution, all instance documents that conform to the existing XML schema are copied to a temporary location in the database, and the existing XML schema is deleted. The modified XML schema is registered, and all the instance documents are inserted into their new location from the temporary area.

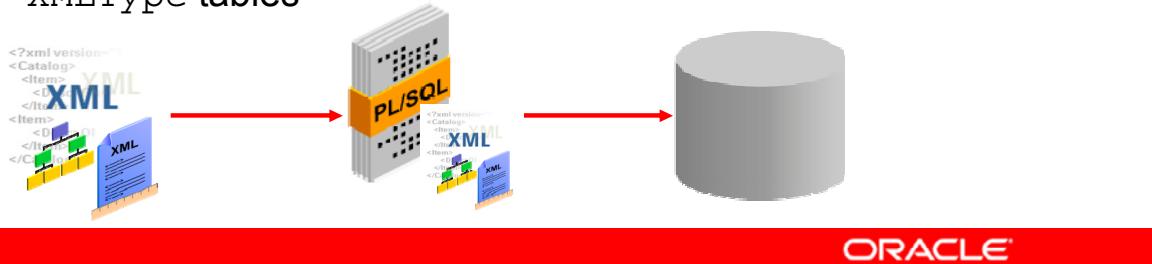
You can use the `DBMS_XMLSHEMA.COPYEVOLVE` PL/SQL procedure to perform copy-based XML schema evolution.

The in-place XML schema evolution has some restrictions that do not apply to the copy-based XML schema evolution. In general, the in-place XML schema evolution is permitted if you do not change the storage model and if the changes do not invalidate existing documents. You see examples of supported operations and restrictions for the in-place XML schema evolution later in this lesson.

Though copy-based XML schema evolution does not have the same restrictions as the in-place XML schema evolution, the process can be time-consuming if there is a large number of associated instance documents.

DBMS_XMLSHEMA.COPYEVOLVE Procedure

- Use the DBMS_XMLSHEMA.COPYEVOLVE procedure to evolve an XML schema.
- You need specific privileges for copy-based XML schema evolution.
- This procedure:
 - Creates a copy of the existing instance documents in the temporary XMLType tables
 - Drops the old version of the XML schema and registers the new version
 - Copies the backed up instance documents in the new XMLType tables



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

You can evolve an XML schema by using the DBMS_XMLSHEMA.COPYEVOLVE procedure.

To perform copy-based XML schema evolution, you need the following:

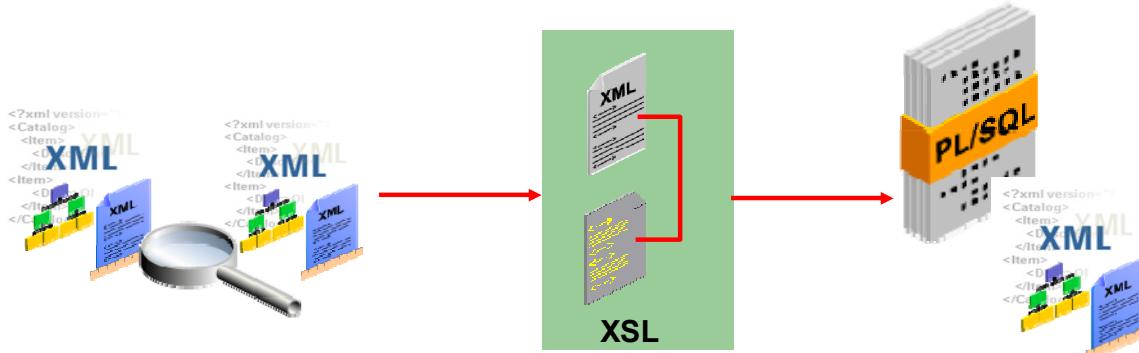
- Type-related privileges: DROP TYPE, CREATE TYPE, and ALTER TYPE
- All privileges on the XMLType tables and the XMLType columns that conform to the XML schemas being evolved

If there are XML schema-based XMLType tables or columns in the database schemas of other users, copy-based schema evolution can be performed by a DBA to avoid granting all privileges to the XML schema owner.

The DBMS_XMLSHEMA.COPYEVOLVE PL/SQL procedure:

- Provides a backup of instance documents by copying them to the temporary XMLType tables
- Drops the old version of the XML schema (along with the associated instance documents) and registers the new version
- Copies the backed-up instance documents to the new XMLType tables (after deleting the old XML schema)

Using DBMS_XMLSHEMA.COPYEVOLVE: Steps



1. Identify the dependent XML schemas.

2. Use the XSL style sheet to transform the existing instance documents.

3. Call the DBMS_XMLSHEMA.COPYEVOLVE procedure.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

1. Identify the XML schemas that are dependent on the XML schema that is to be evolved. You can obtain the URLs of the dependent XML schemas by using the following query:

```
SELECT dxs.SCHEMA_URL
FROM DBA_DEPENDENCIES dd, DBA_XML_SCHEMAS dxs
WHERE dd.REFERENCED_NAME = (SELECT INT_OBJNAME
FROM DBA_XML_SCHEMAS
WHERE SCHEMA_URL = schema_to_be_evolved
AND OWNER = owner_of_schema_to_be_evolved)
AND dxs.OWNER = owner_of_schema_to_be_evolved
AND dxs.INT_OBJNAME = dd.NAME;
```

2. If the existing instance documents do not conform to the new XML schema, you must provide an Extensible Stylesheet Language (XSL) style sheet that, when applied to an instance document, transforms it to conform to the new schema. This must be done for each XML schema identified in step 1. The transformation must handle documents that conform to all the top-level elements in the new XML schema.
3. Call the `DBMS_XMLSCHEMA.COPYEVOLVE` procedure, specifying the XML schema URLs, new schemas, and transformations.

Copy-Based XML Schema Evolution Example: Steps Overview

1. Load the XML schema into the Oracle XML DB Repository and register the schema.
 - code_05_35_s.sql
2. Create the `emp_acme_tab` table with the following columns: `empId`, `comments`, `start_date`, and `employeeshireinfo` (XMLType column).
 - code_05_37_s.sql
3. Insert values into the `emp_acme_tab` table.
 - code_05_38_s.sql
4. Query for the number of rows in the `emp_acme_tab` table.
 - code_05_39_s.sql



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The code examples in the next few slides describe the following scenario, which demonstrates the copy-based XML schema evolution method. This slide and the next describe the overview of the copy-based XML schema.

Copy-Based XML Schema Evolution Example: Steps Overview

5. Add a new address element to the existing XML schema definition.
 - code_05_40_s.sql

[Optional: Try to insert new values in emp_acme_tab.

 - code_05_42_s.sql]
6. Evolve the XML schema , and commit the changes.
 - code_05_41_s.sql
7. Insert new values in the emp_acme_tab table.
 - code_05_42_s.sql
8. Query the emp_acme_tab table to see if the table has both the existing and the newly added data after the schema evolution.
 - code_05_43_as.sql and code_05_43_bs.sql



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Note

- The optional part of step 5 in the slide will fail since the XML schema is not yet evolved and therefore does not contain the newly added element address. This is the same code example as step 7. This optional step will generate an error message. This optional step is not shown in the subsequent pages.
- In step 7, it is important to commit the changes after you evolve the XML schema; otherwise, when you access the XML DB repository, you will not see a new resource created for the new XML schema.

Step 1: Loading the XML Schema into the Oracle XML DB Repository and Registering the Schema

```
-- The complete code example is listed in the notes section of this page
-- and the next full notes page. This is the code_05_35_s.sql script.

DECLARE
    res boolean;
    xmlschema xmltype := xmltype(
    . . .

    IF (DBMS_XDB_REPOS.ExistsResource('/public/employeeDetails.xsd')) THEN
        DBMS_XDB_REPOS.DeleteResource('/public/employeeDetails.xsd');
    END IF;
    res :=
    DBMS_XDB_REPOS.CreateResource('/public/employeeDetails.xsd',xmlschema);
END;
/
BEGIN
    DBMS_XMLSCHEMA.registerSchema
    ('www.employeeinfo.com',xdburiType('/public/employeeDetails.xsd'),TRUE,TRUE,TRUE);
    END;
/

```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The code example in the slide (the complete code is on the next page) loads the employeeDetails.xsd schema definition into the Oracle XML DB Repository and, and then registers the XML schema with the URL www.employeeinfo.com.

The output of the slide example is as follows:

The screenshot shows the Oracle SQL Developer interface with a "Script Output" window. The window title is "Script Output". It contains the following text:
Task completed in 2.654 seconds
Connected
anonymous block completed
anonymous block completed
Connection created by CONNECT script command disconnected

Step 1: Loading the XML Schema into the Oracle XML DB Repository and Registering the Schema

The first screen capture displays the complete code example on the previous slide. The second screen capture shows the employeeDetails.xsd resource in the XML repository.

```
-- Use SQL*Plus for this example as hr/hr
-- This code Loads the XML Schema into the Oracle XML DB Repository and Registers the XML schema.

connect hr/hr
declare
    res boolean;
    xmlschema xmltype := xmltype(
        '<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xdb="http://xmlns.oracle.com/xdb">
            <xsd:complexType name="empl" xdb:SQLType="emp_t">
                <xsd:sequence>
                    <xsd:element name="employeedetails" type="empdetailsType"/>
                    <xsd:element name="acme_info" type="acme_infoType"/>
                    <xsd:element name="packageinfo" type="packageinfoType"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="empdetailsType" xdb:SQLType="empdetails_T">
                <xsd:sequence>
                    <xsd:element name="employee_name" type="xsd:string"/>
                    <xsd:element name="emp_years" type="xsd:decimal"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="acme_infoType" xdb:SQLType="acmeinfo_T">
                <xsd:sequence>
                    <xsd:element name="acme_startdate" type="xsd:date"/>
                    <xsd:element name="acme_standard" type="xsd:decimal"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="packageinfoType" xdb:SQLType="packageinfo_T">
                <xsd:sequence>
                    <xsd:element name="package" type="xsd:decimal"/>
                    <xsd:element name="month" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:element name="employee" type="empl"/>
        </xsd:schema>');
begin
    if (dbms_xdb_repos.existsResource('/public/employeeDetails.xsd')) then
        dbms_xdb_repos.deleteResource('/public/employeeDetails.xsd');
    end if;
    res := dbms_xdb_repos.createResource('/public/employeeDetails.xsd',xmlschema);
end;
/
begin
    dbms_xmlschema.registerSchema ('www.employeefinfo.com',xdburitype('/public/employeeDetails.xsd'),TRUE,TRUE,FALSE,TRUE);
end;
/

```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xdb="http://xmlns.oracle.com/xdb">
    <xsd:complexType name="empl" xdb:SQLType="emp_t">
        <xsd:sequence>
            <xsd:element name="employeedetails" type="empdetailsType"/>
            <xsd:element name="acme_info" type="acme_infoType"/>
            <xsd:element name="packageinfo" type="packageinfoType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="empdetailsType" xdb:SQLType="empdetails_T">
        <xsd:sequence>
            <xsd:element name="employee_name" type="xsd:string"/>
            <xsd:element name="emp_years" type="xsd:decimal"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="acme_infoType" xdb:SQLType="acmeinfo_T">
        <xsd:sequence>
            <xsd:element name="acme_startdate" type="xsd:date"/>
            <xsd:element name="acme_standard" type="xsd:decimal"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="packageinfoType" xdb:SQLType="packageinfo_T">
        <xsd:sequence>

```

Step 2: Creating the emp_acme_tab Table

```
-- The code_05_37_s.sql script.

CREATE TABLE emp_acme_tab
  (empId VARCHAR2(100) CONSTRAINT pk_empacme PRIMARY KEY
   ,comments VARCHAR2(20)
   ,start_date DATE
   ,employeeshireinfo XMLTYPE)
XMLTYPE COLUMN employeeshireinfo
XMLSCHEMA "www.employeeinfo.com" element "employee"
  /
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The code example in the slide creates the emp_acme_tab table with the empId, comments, start_date, and employeeshireinfo (XMLType column) columns.

Step 3: Inserting Values Into the emp_acme Table

```
-- The code_05_38_s.sql script.

INSERT INTO emp_acme_tab(empId ,comments ,start_date
,employeeshireinfo)
VALUES ('A100','HR','12-july-2005',
XMLType(
'<?xml version="1.0" encoding="UTF-8"?>
<employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="www.employeeinfo.com">
<employeedetails>
<employee_name>John</employee_name>
<emp_years>4</emp_years>
</employeedetails>
<acme_info>
<acme_startdate>2004-06-12</acme_startdate>
<acme_standard>3</acme_standard>
</acme_info>
...

```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

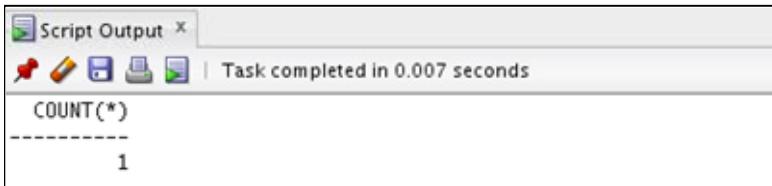
The partial code example in the slide inserts values in the emp_acme table.

The following is the output of the slide example:

The screenshot shows the Oracle SQL Developer interface with the 'Script Output' tab selected. The output window displays the following message:
Task completed in 0.041 seconds
1 rows inserted.

Step 4: Querying for the Number of Rows in the emp_acme Table

```
-- The code_05_39_s.sql script.  
  
SELECT COUNT(*)  
FROM emp_acme_tab;
```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. The title bar says 'Script Output'. Below it, there are icons for redo, undo, save, and others. A status bar at the bottom says 'Task completed in 0.007 seconds'. The main area displays the query results:

COUNT(*)

1



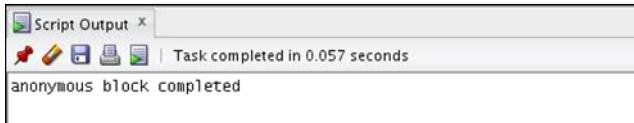
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The code example in the slide queries for the number of rows in the emp_acme table after inserting the values. There is only one row in the table.

Step 5: Adding a New address Element to the Existing XML Schema Definition

```
declare
  xmlschema xmldtype := xdburitype('/public/employeeDetails.xsd').getXML();
  res boolean;
  begin
select XMLQuery('declare namespace xsd="http://www.w3.org/2001/XMLSchema";
copy $1 := $p1 modify (for $j in $i/xsd:schema/xsd:complexType[@name="acme_infoType"]/xsd:sequence
return (#ora:child-element-name xsd:element#) {insert node $p2 into $j}) return $1'
  passing xmlschema as "p1",
  xmldtype('<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="address" type="xsd:string"/>') as "p2"
  returning content)
into xmlSchema
from dual;
| if (dbms_xdb_repos.existsResource('/public/new_employeeDetails.xsd')) then
  dbms_xdb_repos.deleteResource('/public/new_employeeDetails.xsd');
end if;

  res := dbms_xdb_repos.createResource('/public/new_employeeDetails.xsd',xmlschema);
end;
/
```



The Oracle logo, consisting of the word 'ORACLE' in a red sans-serif font.

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

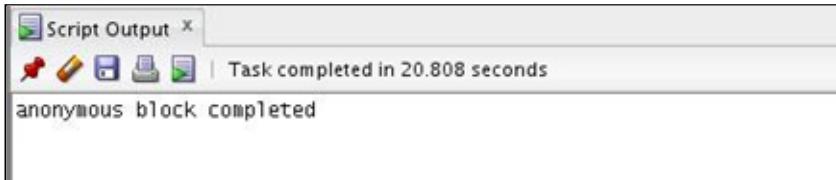
In this step of the example, you add the `address` element to the XML schema definition.

Note: As an optional step, if you attempt to run the `code_04_42_s.sql` script, an error is raised since you have not evolved the XML schema. There is no `address` element in the original definition of the XML schema.

Step 6: Evolving the XML Schema

```
-- The code_05_41_s.sql script. If you don't include the
-- COMMIT, you will not see the new_employeeDetails.xsd
-- resource in the repository.

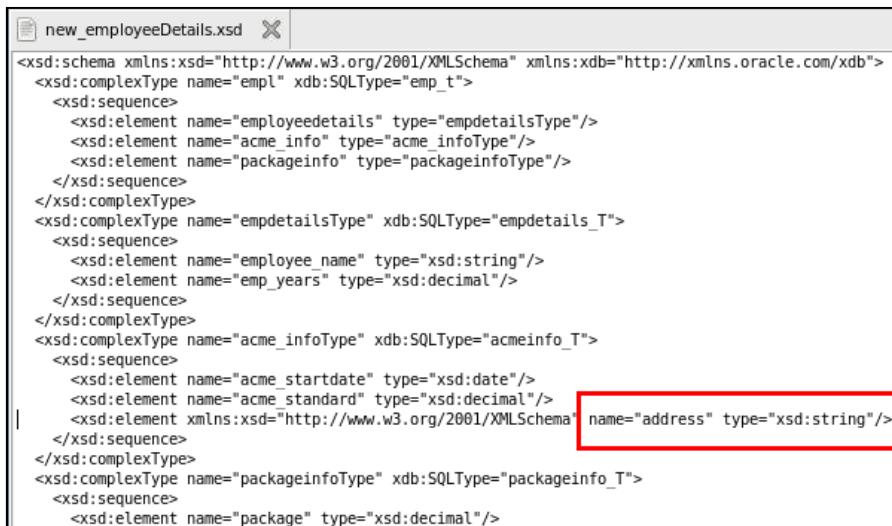
BEGIN
    DBMS_XMLSHEMA.COPYEVOLVE
    (
        xdb$string_list_t('www.employeeinfo.com'),
        XMLSequenceType(xdburiType('/public/new_employeeDetails.xsd')
            .getXML()), null );
COMMIT;
END;
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The code example in the slide shows you how to use the `DBMS_XMLSHEMA.COPYEVOLVE` procedure to evolve the XML schema. The new version of the `new_employeeDetails.xsd` schema definition is specified. Here, because the transformation is not required, an XSL style sheet is not provided. The new address element is added to `new_employeeDetails.xsd`.



Step 7: Inserting New Values in the emp_acme Table

```
-- The code_05_42_s.sql script.

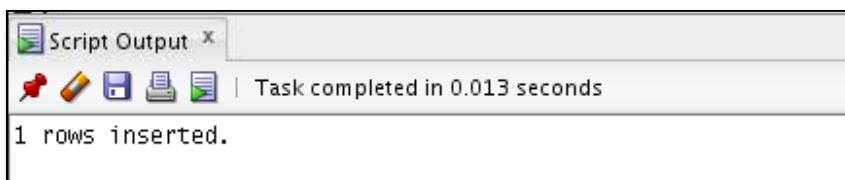
INSERT INTO emp_acme_tab (empId ,comments ,start_date ,employeehireinfo)
values ('A1002','MKTG','14-Nov-2005',
XMLType('<?xml version="1.0" encoding="UTF-8"?>
<employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="www.employeeinfo.com">
<employeedetails>
    <employee_name>Julie </employee_name>
    <emp_years>3</emp_years>
</employeedetails>
<acme_info>
    <acme_startdate>2004-06-12</acme_startdate>
    <acme_standard>3</acme_standard>
    <address>Boston</address>
</acme_info>
<packageinfo>
    <package>2000</package>
    <month> july</month>
</packageinfo>
</employee>'))
/
{
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In this step of the example, you add a second row of data into the emp_acme_tab table. The employeehireinfo XMLType column for empID A1002 contains the new address element.

The output of the slide example is as follows:



Step 8: Querying the emp_acme Table to See the Changes

```
-- The code_05_43_as.sql script.
```

```
SELECT empid, XMLCast (XMLQuery('/employee/acme_info/address'
    PASSING employeeshireinfo
    RETURNING CONTENT) AS VARCHAR2(100)) "ADDRESS"
FROM emp_acme_tab;
```

EMPID	ADDRESS
1 A100	(null)
2 A1002	Boston

```
-- The code_05_43_bs.sql script.
```

```
SELECT count(*)
FROM emp_acme_tab;
```

COUNT(*)

2



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Now that the EMP_ACME_TAB table has the latest data, you can query and view the result as shown in the first code example in the slide.

You can also use the second code example to verify the number of rows in the table which is now 2.

Oracle XML DB copies the data in the EMP_ACME_TAB table to the temporary tables. It then drops the emp_acme_tab table and deletes the old XML schema definition. After registering the new XML schema definition, it creates the EMP_ACME_TAB table and populates the table with the data. You thus see two rows in the EMP_ACME_TAB table. The second row has the address element value Boston.

Note: You can access the content of the documents stored in the Oracle XML DB Repository by using XDBURIType. XDBURIType, which uses a URL to specify which resource to access. The URL passed to XDBURIType is assumed to start at the root of the Repository. The XDBURIType data type provides the getBLOB(), getCLOB(), and getXML() methods to access the different kinds of content that can be associated with a resource.

Copy-Based XML Schema Evolution: Guidelines

When evolving an XML schema by using DBMS_XMLSHEMA.COPYEVOLVE, follow these guidelines:

- Ensure that the temporary tables holding the old documents are not dropped.
- Ensure that there are no concurrent sessions using the XML schema and its dependents.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If top-level elements have been dropped or their names have been changed, you can call the DBMS_XMLSHEMA.copyEvolve procedure with the genTables parameter set to FALSE and the preserveOldDocs parameter set to TRUE. This ensures that the temporary tables holding the old documents are not dropped and that new tables are generated.

If there are concurrent sessions with shared locks on an XML schema, the DBMS_XMLSHEMA.copyEvolve procedure waits for these sessions to release the locks. You should thus ensure that, during the schema evolution process, there are no concurrent sessions using the XML schema and its dependents.

Note: DBMS_XMLSHEMA.copyEvolve either completely succeeds or raises an error, in which case it attempts to roll back as much of the operation as possible. Evolving a schema involves many database data definition language (DDL) statements. When an error occurs, the compensating DDL statements are executed to undo the effect of all the steps executed to that point. In certain cases, you cannot roll back the operation. For example, if table creation fails due to reasons that are not related to the new schema (for example, from insufficient privileges), there is no way to roll back. The temporary tables are not deleted even if preserveOldDocs is false; as a result, the data can be recovered.

Using DBMS_XMLSHEMA.COPYEVOLVE: Disadvantages

DBMS_XMLSHEMA.COPYEVOLVE has the following disadvantages:

- There is additional overhead due to XSL transformation.
- Indexes, triggers, and constraints need to be re-created after XML schema evolution.
- Top-level element names may be changed.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Copy-based XML schema evolution requires that the copy operations move the data from the old infrastructure to the new infrastructure. If the original documents are not compliant with the new XML schema, the XSL transformation that is required to convert the documents is an additional overhead to the process.

In the process of XML schema evolution, the metadata of the dependent XMLType tables is not preserved. You, therefore, need to re-create indexes, triggers, and constraints after the evolution.

With copy-based XML schema evolution, top-level element names might be changed. You must ensure that, at the end of the procedure, new tables are generated and the temporary tables holding the old instance documents are not dropped.

In-Place XML Schema Evolution

- In-place XML schema evolution makes changes to an XML schema without requiring that the existing data be copied, deleted, and reinserted.
- The set of operations that are permitted in in-place schema evolution is smaller than what is permitted in copy-based evolution.



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

By using XML schema evolution, you can make changes to an XML schema that is registered in Oracle XML DB, and that may have instance XML documents.

With in-place XML schema evolution, you can evolve an XML schema without requiring a copy of the existing instance documents. The set of operations that are permitted in in-place schema evolution is smaller than what is permitted in copy-based evolution. In-place XML schema evolution is faster than copy-based evolution.

The primary restriction on using in-place evolution can be stated generally as a requirement that a given XML schema can be evolved in place in only a backward-compatible way. Backward-compatible here means that any possible instance document that would validate against a given XML schema must also validate against a later (evolved) version of that XML schema.

Note: For additional information about the restrictions for in-place XML schema evolution, see the *Oracle XML DB Developer's Guide 12c Release 2 (11.2)* reference guide.

In-Place XML Schema Evolution: Advantages

In-place XML schema evolution:

- Avoids data copy
- Takes minimal time



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In copy-based schema evolution, all instance documents that conform to the XML schema are copied to a temporary location in the database, the old schema is deleted, the modified schema is registered, and then the instance documents are inserted into their new locations from the temporary area. During copy-based XML schema evolution, database objects, such as XML schemas, XML tables, and so on, are potentially inaccessible to you for a long time. The time taken to complete copy-based XML schema evolution is directly proportional to the amount of data being managed. Oracle Database addresses the issues with copy-based XML schema evolution by offering in-place XML schema evolution. The advantage with in-place XML schema evolution is that it does not unload and reload the data in the XMLType tables and columns that are bound to the XML schema being modified. Therefore, the time taken by in-place XML schema evolution is minimal and constant, and is not dependent on the amount of data being managed.

The requirement with in-place XML schema evolution is that a given XML schema can be evolved in place only in a backward-compatible way. This means that any possible instance document that would validate against a given XML schema must also validate against a later evolved version of that XML schema.

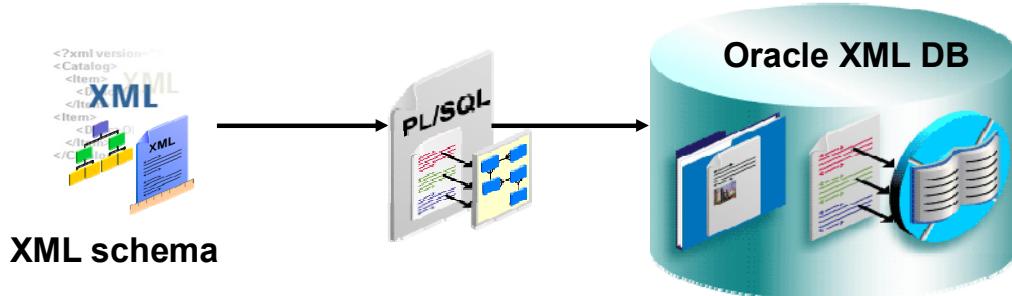
In general, you can use in-place XML schema evolution if you are not changing the storage model and if the changes do not invalidate existing documents.

Performing In-Place XML Schema Evolution

- Use the DBMS_XMLSHEMA.INPLACEEVOLVE PL/SQL procedure to perform in-place XML schema evolution.
- Syntax:

```
DBMS_XMLSHEMA.INPLACEEVOLVE(schemaURL IN VARCHAR2,  
                                diffXML IN XMLType, flags IN NUMBER);
```

- INPLACEEVOLVE raises errors for invalid XPath expressions and invalid changes to an XML schema.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

DBMS_XMLSHEMA.INPLACEEVOLVE Procedure

The DBMS_XMLSHEMA.INPLACEEVOLVE procedure evolves a registered XML schema by propagating the schema changes to object types and tables. You can specify an XML schema-differences document and (optionally) specify flags to be applied to the evolution process. The following is the syntax description:

- schemaURL: URL of the XML schema to be evolved
- diffXML: Changes to be applied to the XML schema. This is an XML document that conforms to the `xdiff` XML schema. It specifies what changes need to be applied and the locations in the XML schema document where the changes are to be applied.

- **flags:** A bit mask that controls the behavior of the procedure. You can set the following bit values in this mask:
 - **INPLACE_EVOLVE** (value 1, which means that bit 1 is on): Perform in-place XML schema evolution. In-place evolution constructs a new version of an XML schema by applying the changes specified in a `diffXML` document, validates that new XML schema (against the XML schema for XML schemas), constructs DDL statements to evolve the disk structures that are used to store the XML instance documents associated with the XML schema, executes these DDL statements, and replaces the old version of the XML schema with the new, in that order.
 - **INPLACE_TRACE** (value 2, which means that bit 2 is on): Perform all the steps that are necessary for in-place evolution, except the following: executing the DDL statements and overwriting the old XML schema with the new. Then, write both the DDL statements and the new XML schema to a trace file.

The `DBMS_XMLSHEMA.inPlaceEvolve` procedure raises an error in the following cases:

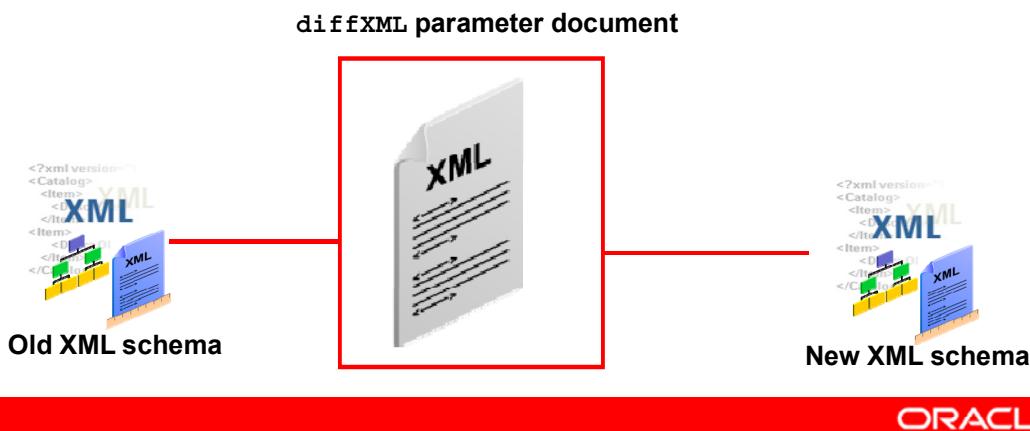
- An XPath expression is invalid.
- The `diffXML` document does not conform to the `xdiff` XML schema.
- An XPath expression is syntactically correct, but does not target a node in the XML schema.
- The changes make the XML schema invalid or not well formed.
- A generated DDL statement, such as `CREATE TYPE` or `ALTER TYPE`, causes a problem on execution.

Note: Bit `INPLACE_EVOLVE` means carry out the evolution steps and replace the old XML schema with the new. Bit `INPLACE_TRACE` means save the evolution steps and the new XML schema in a trace file but do not carry out the evolution steps.

Creating an XML Document for the `diffXML` Parameter

The `diffXML` parameter document:

- Contains a sequence of operations that describes the changes between the old XML schema and the new XML schema
- Must conform to the `xdiff` XML schema



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You must create the XML document to be used for the `diffXML` parameter. The value of the `diffXML` parameter is passed to the `DBMS_XMLSHEMA.inPlaceEvolve` procedure. This is an XML document (as an `XMLType` instance) that specifies the changes to be applied to an XML schema for in-place XML schema evolution. The changes specified by the `diffXML` document are applied in order. You can use any of the following methods to create an XML document to be used for the `diffXML` parameter:

- The `XMLDiff` JavaBean (`oracle.xml.differ.XMLDiff`)
- The `xmldiff` command-line utility
- The `XMLDiff` SQL function

For more information about the `xmldiff` command-line utility and the `XMLDiff` JavaBean, refer to the *Oracle XML Developer's Kit Programmer's Guide*.

`xdiff.xsd` is the Oracle XML DB-supplied XML schema to which the document specified as the `diffXML` parameter to procedure `DBMS_XMLSHEMA.inPlaceEvolve` must conform.

Note: To simplify the task of comparing two versions of XML documents to see what has changed, Oracle Database 12c provides the `XMLDIFF()` SQL function.

In-Place XML Schema Evolution: Example

1. Create `purchaseOrder.v2.xsd` from `purchaseOrder.xsd`.
2. Insert into the `PURCHASEORDER` table an XML document that conforms to `purchaseOrder.v2.xsd`.
3. Perform in-place XML schema evolution.
4. Repeat step 2.
5. Query the `PURCHASEORDER` table to verify the successful insertion of the new XML document.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The code examples for the steps in the slide are available on the following pages.

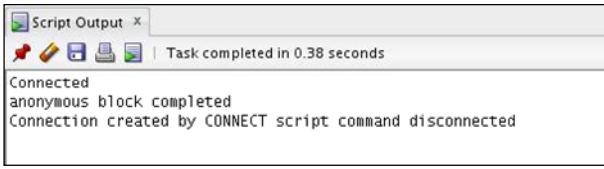
In-Place XML Schema Evolution Example: Step 1

```
connect oe/oe;
declare
  new_schema xmltype;
  res boolean;
begin
  if (dbms_xdb_repos.existsResource('/home/0E/purchaseOrder.v2.xsd')) then
    dbms_xdb_repos.deleteResource('/home/0E/purchaseOrder.v2.xsd');
  end if;
select XMLQuery('declare namespace xs="http://www.w3.org/2001/XMLSchema"; declare namespace xdb="http://xmlns.oracle.com/xdb"; copy $1 := $p1 modify
  (for $j in $1/xs:schema/xs:complexType[@name="LineItemType"]/xs:sequence return insert nodes $p2 as last into $j) return $1'
  passing xdbURIType('/home/0E/purchaseOrder.xsd').getXML() as "p1",
        XMLType('<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" name="Unit" type="xs:string" minOccurs="0" />') as "p2"
  returning content)
  into new_schema
  from dual;

res := dbms_xdb_repos.createResource('/home/0E/purchaseOrder.v2.xsd',new_schema);

commit;

end;
/
```



The Oracle logo, consisting of the word 'ORACLE' in a red sans-serif font with a registered trademark symbol.

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In step 1, you create a new version of an XML schema from an existing XML schema. The slide example creates `purchaseOrder.v2.xsd` by making changes to the existing `purchaseOrder.xsd`.

In-Place XML Schema Evolution Example: Step 2

```

declare
  new_xml xmldtype;
  res boolean;
begin
  select column_value into new_xml
  from xmltable('for $1 in fn:collection("oradb:/OE/PURCHASEORDER")
    where $1/PurchaseOrder/Reference = "SBELL-2002100912333601PDT"
    return $1');

  select XMLQuery('copy $1 := $p1 modify
    (for $j in $1/PurchaseOrder/LineItems/LineItem[@ItemNumber="1"]
      return insert nodes $p2 as last into $j) return $1'
    passing new_xml as "p1", xmldtype('<Unit>Box</Unit>') as "p2"
    returning content)
  into new_xml
  from dual;

  select XMLQuery('copy $1 := $p1 modify
    (for $j in $1/PurchaseOrder/LineItems/LineItem[@ItemNumber="2"]
      return insert nodes $p2 as last into $j) return $1'
    passing new_xml as "p1", xmldtype('<Unit>Carton</Unit>') as "p2"
    returning content)
  into new_xml
  from dual;

  select XMLQuery('copy $1 := $p1 modify
    (for $j in $1/PurchaseOrder/LineItems/LineItem[@ItemNumber="3"]
      return insert nodes $p2 as last into $j) return $1'
    passing new_xml as "p1", xmldtype('<Unit>Case</Unit>') as "p2"
    returning content)
  into new_xml
  from dual;

  select XMLQuery('copy $1 := $p1 modify
    (for $j in $1/PurchaseOrder/Reference return replace value of node $j with $p2) return $1'
    passing new_xml as "p1", 'SBELL-2002100912333601PDT-V2' as "p2"
    returning content)
  into new_xml
  from dual;

  insert into PURCHASEORDER values new_xml;

  commit;
end;
/

```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In step 2 of the example, you try to insert a new XML document that conforms to the new version of the XML schema. Observe the following error:

The screenshot shows the Oracle SQL Developer interface with the 'Script Output' tab selected. The output window displays the following error message:

```

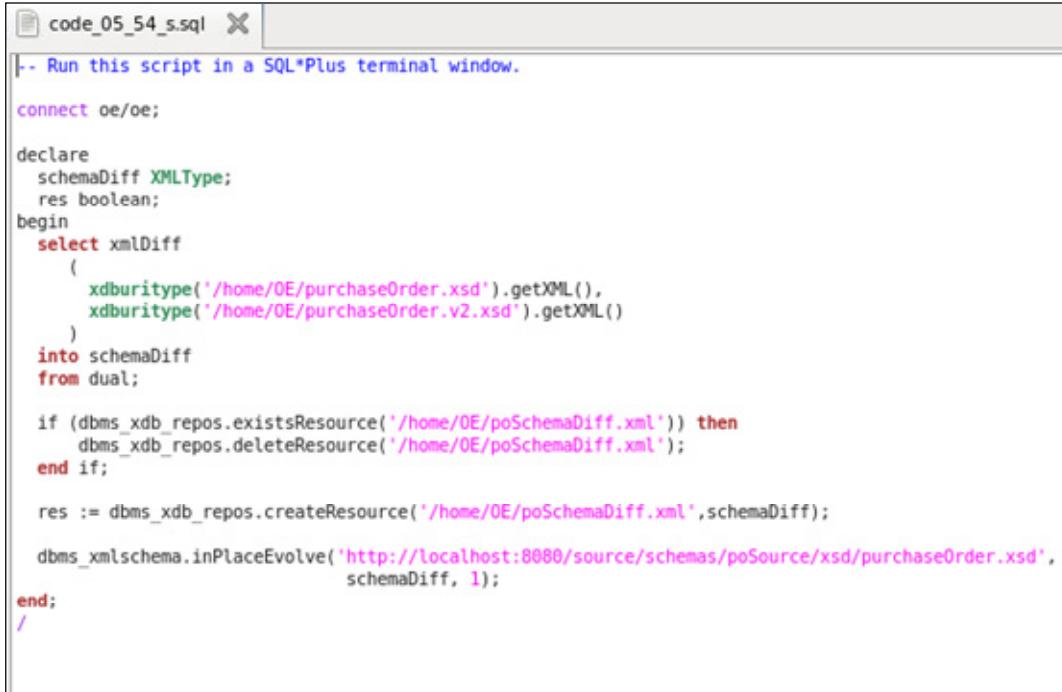
insert into PURCHASEORDER values new_xml;

commit;

end;
Error report:
ORA-30937: No schema definition for 'Unit' (namespace '###local') in parent '/PurchaseOrder/LineItems/LineItem[1]'
ORA-06512: at line 43
30937. 00000 -  "No schema definition for '%s' (namespace '%s') in parent '%s'"
*Cause:   The schema definition for the parent node being processed does
          not allow for the specified child node in its content model.
          Note that any typecasting via xsi:type must occur before the
          schema definitions for the new type can be used.
*Action:  Only insert elements and attributes declared in the schema.
          Check to make sure that xsi:type (if used) is specified first.

```

In-Place XML Schema Evolution Example: Step 3



```
-- Run this script in a SQL*Plus terminal window.

connect oe/oe;

declare
  schemaDiff XMLType;
  res boolean;
begin
  select xmlDiff
  (
    xdburitype('/home/0E/purchaseOrder.xsd').getXML(),
    xdburitype('/home/0E/purchaseOrder.v2.xsd').getXML()
  )
  into schemaDiff
  from dual;

  if (dbms_xdb_repos.existsResource('/home/0E/poSchemaDiff.xml')) then
    dbms_xdb_repos.deleteResource('/home/0E/poSchemaDiff.xml');
  end if;

  res := dbms_xdb_repos.createResource('/home/0E/poSchemaDiff.xml',schemaDiff);

  dbms_xmlschema.inPlaceEvolve('http://localhost:8080/source/schemas/poSource/xsd/purchaseOrder.xsd',
                                schemaDiff, 1);
end;
/
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In step 3 of the example, you use the DBMS_XMLSHEMA.INPLACEEVOLVE procedure.

In-Place XML Schema Evolution Example: Steps 4 and 5

```

declare
  new_xml xmldtype;
  res boolean;
begin
  select column_value into new_xml
  from XMLTable('for $i in fn:collection("oradb:/OE/PURCHASEORDER")
    where $i/PurchaseOrder/Reference = "SBELL-2002100912333601PDT"
    return $i');

  select XMLQuery('copy $1 := $p1 modify
    (for $j in $1/PurchaseOrder/LineItems/LineItem[@ItemNumber="1"]
      return insert nodes $p2 as last into $j) return $1'
    passing new_xml as "p1", xmldtype('<Unit>Box</Unit>') as "p2"
    returning content)
  into new_xml
  from dual;

  select XMLQuery('copy $1 := $p1 modify
    (for $j in $1/PurchaseOrder/LineItems/LineItem[@ItemNumber="2"]
      return insert nodes $p2 as last into $j) return $1'
    passing new_xml as "p1", xmldtype('<Unit>Cartone</Unit>') as "p2"
    returning content)
  into new_xml
  from dual;

  select XMLQuery('copy $1 := $p1 modify
    (for $j in $1/PurchaseOrder/LineItems/LineItem[@ItemNumber="3"]
      return insert nodes $p2 as last into $j) return $1'
    passing new_xml as "p1", xmldtype('<Unit>Case</Unit>') as "p2"
    returning content)
  into new_xml
  from dual;

  select XMLQuery('copy $1 := $p1 modify
    (for $j in $1/PurchaseOrder/Reference return replace value of node $j with $p2) return $1'
    passing new_xml as "p1", 'SBELL-2002100912333601PDT-V2' as "p2"
    returning content)
  into new_xml
  from dual;

  insert into PURCHASEORDER values new_xml;
  commit;
end;
/

```

```

select column_value
from XMLTable ('for $i in fn:collection("oradb:/OE/PURCHASEORDER")
where exists($i/PurchaseOrder/LineItems/LineItem/Unit) return $i/PurchaseOrder/Reference');

```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Step 4: Repeat step 2. Observe that there are no errors this time.



Step 5: Verify that the new XML document is successfully inserted:

```

SELECT column_value
FROM
XMLTable ('for $i in fn:collection("oradb:/OE/PURCHASEORDER")
WHERE exists($i/PurchaseOrder/LineItems/LineItem/Unit) return
$i/PurchaseOrder/Reference');

```

COLUMN_VALUE
<Reference>SBELL-2002100912333601PDT -V2</Reference>

Using In-Place XML Schema Evolution: Guidelines

- Perform the following tasks before you use in-place XML schema evolution:
 - Back up all the existing data for the XML schema that is to be evolved.
 - Perform a dry run by using only trace.
- After you perform an in-place XML-schema evolution:
 - If you are accessing the database using a client that caches data, restart your client. Otherwise, the pre-evolution version of the XML schema might continue to be used locally, with unpredictable results.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

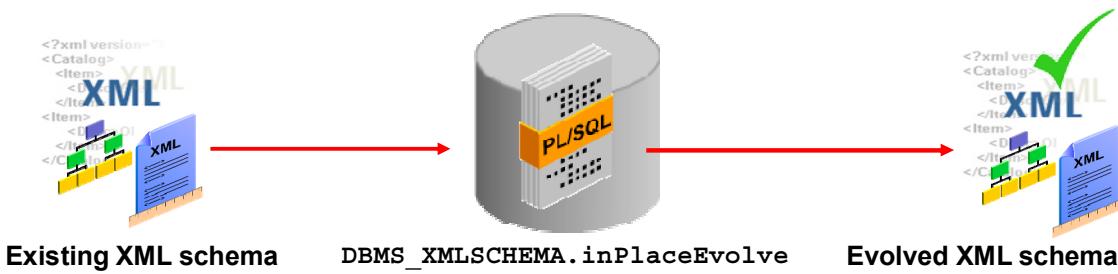
Before you use in-place XML schema evolution, perform the following:

- Back up all existing data (instance documents) for the XML schema that is to be evolved. This is necessary because rollback is not possible after an in-place evolution. If any errors occur during evolution, or if you make a major mistake and need to redo the entire operation, you must be able to go back to the original data.
- Perform a dry run by using only trace. That is, without actually evolving the XML schema or updating any instance documents, produce a trace of the update operations that would be performed during evolution. To do this, set the `flag` parameter value only to `INPLACE_TRACE`. Do not also use `INPLACE_EVOLVE`. After performing the dry run, examine the trace file and verify that the listed DDL operations are those that you intend.
- After you perform an in-place XML-schema evolution, if you are accessing the database by using a client that caches data, or if you are not sure whether this is the case, then restart your client. Otherwise, the pre-evolution version of the XML schema might continue to be used locally, with unpredictable results.

In-Place XML Schema Evolution: Supported Operations

Examples:

- Adding an optional element to a complex type or group
- Converting an element from simple type to complex type with simple content
- Increasing the value attribute of an existing maxLength element
- Adding a global attribute



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The following are some of the operations that are supported for in-place XML schema evolution:

- **Adding an optional element to a complex type or group:** This operation is always permitted. The following is an example of adding the optional `shipmethod` element to the complex type definition.


```
<xs:complexType name="ShippingInstructionsType">
  <xs:sequence>
    <xs:element name="name" type="NameType" minOccurs="0" />
    <xs:element name="address" type="AddressType" minOccurs="0" />
    <xs:element name="telephone" type="TelephoneType"
      minOccurs="0" />
    <xs:element name = "shipmethod" type = "xs:string" minOccurs =
      "0"/>
  </xs:sequence>
</xs:complexType>
```
- **Converting an element from simple type to complex type with simple content:** This operation is supported only if the storage model is binary XML.

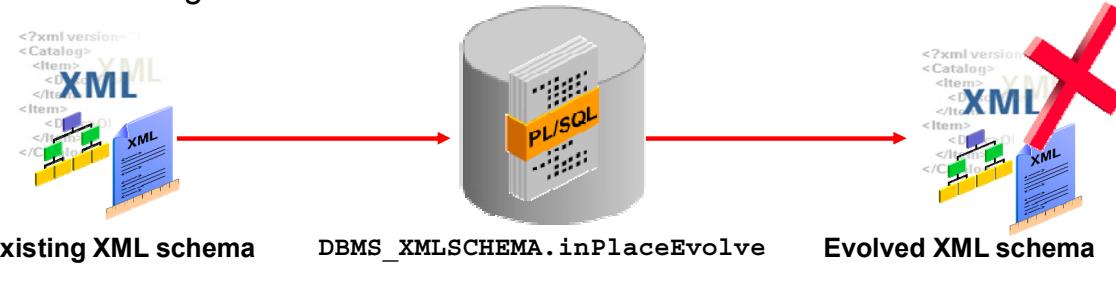
- **Modifying the value attribute of an existing maxLength element:** This operation is always permitted. The value can only be increased, but not decreased.
- **Adding a global attribute:** This operation is always permitted.
- **Changing the value of the maxOccurs attribute:** The value of maxOccurs can only be increased, and this is possible only for data stored as binary XML. That is, you cannot make changes to the maxOccurs attribute for data that is stored object-relationally.
- **Adding an enumeration value:** You can add a new enumeration value only to the end of an enumeration list.
- **Adding a global element:** This operation is always permitted.
- **Adding, modifying, or deleting a comment or processing instruction:** This operation is always permitted.

For more information about the supported operations for in-place XML schema evolution, see the *Oracle XML DB Developer's Guide 12c Release 2 (12.1)*.

In-Place XML Schema Evolution: Restrictions

The following restrictions exist when using in-place XML schema evolution:

- Backward-compatibility restrictions:
 - Changes in disk data layout
 - Reordering of XML schema constructs
 - Changes from collection to noncollection
- Other restrictions:
 - Changes to attributes in the `xdb` namespace
 - Changes from a noncollection to collection



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In-place XML schema evolution avoids data copy and instance document transformation. Therefore, it does not permit arbitrary changes to an XML schema.

A given XML schema can be evolved in place in only a backward-compatible way. In the case of binary XML, backward compatibility also means that any XML schema annotations that affect the binary XML treatment must not change during the process of XML schema evolution.

- **Changes in disk data layout:** Certain changes to an XML schema alter the layout of the corresponding instance documents on disk. Therefore, they are not permitted by in-place XML schema evolution. This situation is more common when the storage layer is tightly integrated with the information derived from the XML schema, as is the case with object-relational storage.

An example of altering the disk data layout is an XML schema that is registered for object-relational storage mapping. This XML schema is evolved by splitting a complex type into two complex types.

Example: The `ShippingInstructionsType` complex type is split into two complex types, `Person-Name` and `Contact-Info`, and the `ShippingInstructionsType` complex type is deleted.

Although the XML schema being evolved may have no instance documents, and therefore no data copy is required, a change in the layout of the existing tables is required to accommodate future instance documents.

```
<complexType name="ShippingInstructionsType">
<sequence>
<element name="name" type="NameType" minOccurs="0"/>
<element name="address" type="AddressType" minOccurs="0"/>
<element name="telephone" type="TelephoneType" minOccurs="0"/>
</sequence>
</complexType>
<complexType name="Person-Name">
<sequence>
<element name="name" type="NameType" minOccurs="0"/>
</sequence>
</complexType>
<complexType name="Contact-Info">
<sequence>
<element name="address" type="AddressType" minOccurs="0"/>
<element name="telephone" type="TelephoneType" minOccurs="0"/>
</sequence>
</complexType>
```

- **Reordering of XML schema constructs:** You cannot use in-place evolution to reorder schema elements in a way that affects the Document Object Model (DOM) fidelity of the instance documents. For example, you cannot change the order of elements within a `<sequence>` element in a complex type definition.

Example: A complex type named `ShippingInstructionsType` requires that the `name`, `address`, and `telephone` of its child element be in that order. You cannot use in-place evolution to change the order to `name`, `telephone`, and `address`.

- **Changes from a collection to a noncollection:** You cannot use in-place evolution to change a collection to a noncollection (for example, changing from a `maxOccurs` value greater than one to a `maxOccurs` value of one).

The following are some of the restrictions on in-place XML schema evolution other than the backward compatibility restrictions.

- **Changes to attributes in the `xdb` namespace:** Oracle XML DB uses a set of attributes in the `http://xmlns.oracle.com/xdb` namespace. These attributes dictate certain aspects of the object SQL type to be used and the memory layout to be used. Except for the `xdb:defaultTable` attribute, you cannot use in-place evolution to modify attributes in the `http://xmlns.oracle.com/xdb` namespace (with the predefined `xdb` prefix).
- **Changes from a noncollection to a collection:** When XML data is stored object-relationally, you cannot use in-place XML schema evolution to change a noncollection object type to a collection object type. An example is adding an element to a complex type with the element name matching the name of an element already present in the type (or in another type that is related to the first type through inheritance).

Quiz

The XML schema evolution:

- a. Can be used to manage changes to an XML schema
- b. Is useful when the structure or format of XML documents must be changed to reflect the changes in application requirements
- c. Supports three kinds of XML schema evolution:
 - In-place XML schema evolution
 - Copy-based XML schema evolution
 - `diffXML` schema evolution



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a, b

Summary

In this lesson, you should have learned how to:

- Describe the XML schema support in Oracle XML DB
- Create XML schema-based XMLType tables
- Register an XML schema in Oracle XML DB
- Delete a registered XML schema
- Manage changes in an XML schema
- Perform copy-based and in-place XML schema evolution



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 5-3: Overview

This practice covers managing changes in an XML schema by using DBMS_XMLSHEMA.copyEvolve.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In this practice, you manage the changes in an XML schema by using the XML schema evolution technique that is available in Oracle XML DB.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Oracle XML DB Manageability



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Identify why and how to annotate XML schemas
- Review the Purchase-Order, purchaseOrder.xsd, XML schema
- Annotate an XML schema by using the PL/SQL package DBMS_XMLSHEMA_ANNOTATE
- Review some of the Oracle XML DB XML schema elements annotations
- Identify XML schema annotation guidelines for object-relational storage
- Review the DBMS_XMLSTORAGE_MANAGE PL/SQL package



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Agenda

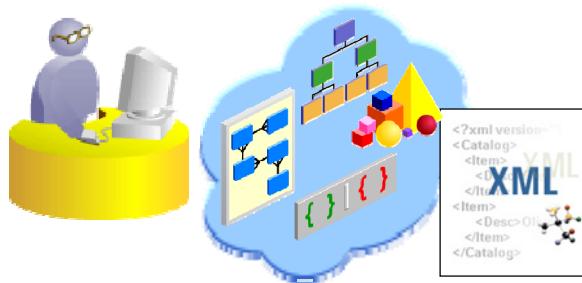
- Oracle XML schema annotations
 - Methods for adding annotations
 - Reviewing the Purchase-Order XML schema
 - Use the DBMS_XMLSHEMA_ANNOTATE PL/SQL package to annotate an XML schema
 - Oracle XML DB XML schema elements annotations examples
- Using the DBMS_XMLSTORAGE_MANAGE PL/SQL package to manage and modify XML storage



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle XML Schema Annotations

- You can annotate XML schemas to influence the objects and tables that are generated by the XML schema registration process.
- You annotate XML schemas by adding Oracle-specific attributes to `complexType`, `element`, and `attribute` definitions that are declared by the XML schema.
- Schema annotations are relevant only for the object-relational storage model.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The W3C XML Schema Recommendation defines an annotation mechanism that lets vendor-specific information be added to an XML schema. Oracle XML DB uses this mechanism to control the mapping between the XML schema and database features.

- You can annotate XML schemas to influence the objects and tables that are generated by the XML schema registration process.
- You annotate XML schemas by adding Oracle-specific attributes to `complexType`, `element`, and `attribute` definitions that are declared by the XML schema.
- Schema annotations are relevant only for the object-relational storage model. They do not apply to the binary XML storage model.

Most XML attributes used by Oracle XML DB belong to the namespace

`http://xmlns.oracle.com/xdb`. XML attributes used for encoding XML data as binary XML belong to the namespace `http://xmlns.oracle.com/2004/CSX`. To simplify the process of annotating an XML schema, Oracle recommends that you declare namespace prefixes in the root element of the XML schema.

Common Uses of XML Schema Annotations

Some of the common reasons to annotate an XML Schema:

- To control the names of the tables, objects, and object attributes created by the `registerSchema` procedure
 - Set the `GENTYPES` or `GENTABLES` parameter to `TRUE`
- To prevent the generation of mixed-case names that require the use of quoted identifiers when working with SQL
- To allow XPath rewrite for object-relational storage for document-correlated recursive XPath queries



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Some of the common reasons to annotate an XML Schema include the following:

- To ensure that the names of the tables, objects, and object attributes created by PL/SQL procedure `DBMS_XMLSCHEMA.registerSchema` for object-relational storage of `XMLType` data are easy to recognize and compliant with any application-naming standards. To accomplish this requirement:
 - Set the values of the `GENTYPES` or `GENTABLES` parameter to `TRUE`.
 - `TRUE` is the default value for the two parameters.
- To prevent the generation of mixed-case names that require the use of quoted identifiers when working directly with SQL.
- To allow XPath rewrite for object-relational storage for document-correlated recursive XPath queries. This rewrite applies to certain applications of SQL/XML access and query functions whose XQuery-expression argument targets recursive XML data.

Note: You do not need to specify values for any of these attributes. Oracle XML DB provides appropriate values by default during the XML schema registration process. However, if you are using object-relational storage, then Oracle recommends that you specify the names of at least the top-level SQL types, so that you can reference them later.

Annotations Methods

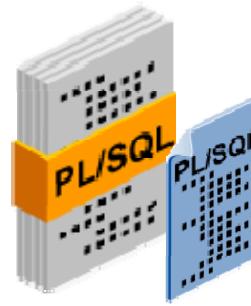
You can add annotations:

- Manually by editing the XML schema document
- Programmatically by invoking annotation-specific subprograms of the DBMS_XMLSHEMA_ANNOTATE package



```
<?xml version="">
<Catalog>
  <Item>
    <Desc>Oil</Desc>
  </Item>
</Catalog>
```

Edit manually by using
an XML editor



DBMS_XMLSHEMA_ANNOTATE
package subprograms

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can author and edit XML schemas manually using any of the following methods:

- A simple text editor, such as `gedit`, `emacs`, or `vi`. You can also use an XML schema-aware editor, such as the XML editor included with Oracle JDeveloper
- An explicit XML schema-authoring tool, such as `XMLSpy` from Altova Corporation

For the most common annotations, you can invoke the DBMS_XMLSHEMA_ANNOTATE annotation-specific PL/SQL subprograms. For more information, see "DBMS_XMLSHEMA_ANNOTATE" in the *Oracle Database PL/SQL Packages and Types Reference 12c Release 1 (12.1)* documentation guide.

Purchase-Order XML Schema, purchaseOrder.xsd, Before the Annotations

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
<xs:element name="PurchaseOrder" type="PurchaseOrderType">
<xs:complexType name="PurchaseOrderType">
    <xs:sequence>
        <xs:element name="Reference" type="ReferenceType"/>
        <xs:element name="Actions" type="ActionsType"/>
        <xs:element name="Reject" type="RejectionType" minOccurs="0"/>
        <xs:element name="Requestor" type="RequestorType"/>
        <xs:element name="User" type="UserType"/>
        <xs:element name="CostCenter" type="CostCenterType"/>
        <xs:element name="ShippingInstructions" type="ShippingInstructionsType"/>
        <xs:element name="SpecialInstructions" type="SpecialInstructionsType"/>
        <xs:element name="LineItems" type="LineItemsType"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="LineItemsType">
    <xs:sequence>
        <xs:element name="LineItem" type="LineItemType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="LineItemType">
    <xs:sequence>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide screen capture shows the partial purchaseOrder.xsd XML schema file. The purchase order XML schema demonstrates some key features of a typical XML document:

- Global element PurchaseOrder is an instance of the complexType PurchaseOrderType.
- PurchaseOrderType defines the set of nodes that make up a PurchaseOrder element in that order: Reference, Actions, Reject, Requestor, User, CostCenter, ShippingInstructions, SpecialInstructions, and LineItems.
- LineItems element consists of a collection of LineItem elements.
- Each LineItem element consists of two elements: Description and Part.
- Part element has Id, Quantity, and UnitPrice attributes.

You can control how elements are to be used in documents with indicators. The `<sequence>` indicator is used within a `<complexType>` to specify the exact order of the child elements. The occurrence indicators are used to define how often an element can occur. The `<maxOccurs>` indicator specifies the maximum number of times an element can occur. The `<minOccurs>` indicator specifies the minimum number of times an element can occur. To allow an element to appear an unlimited number of times, use the `maxOccurs="unbounded"` statement.

Annotated Purchase-Order XML Schema, purchaseOrder.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xdb="http://xmlns.oracle.com/xdb"
            version="1.0">
  <xs:element name="PurchaseOrder" type="PurchaseOrderType" xdb:defaultTable="PURCHASEORDER"/>
  <xs:complexType name="PurchaseOrderType" xdb:SQLType="PURCHASEORDER_T">
    <xs:sequence>
      <xs:element name="Reference" type="ReferenceType" minOccurs="1" xdb:SQLName="REFERENCE"/>
      <xs:element name="Actions" type="ActionsType" xdb:SQLName="ACTIONS"/>
      <xs:element name="Reject" type="RejectionType" minOccurs="0" xdb:SQLName="REJECTION"/>
      <xs:element name="Requestor" type="RequestorType" xdb:SQLName="REQUESTOR"/>
      <xs:element name="User" type="UserType" minOccurs="1" xdb:SQLName="USERID"/>
      <xs:element name="CostCenter" type="CostCenterType" xdb:SQLName="COST_CENTER"/>
      <xs:element name="ShippingInstructions" type="ShippingInstructionsType"
                   xdb:SQLName="SHIPPING_INSTRUCTIONS"/>
      <xs:element name="SpecialInstructions" type="SpecialInstructionsType"
                   xdb:SQLName="SPECIAL_INSTRUCTIONS"/>
      <xs:element name="LineItems" type="LineItemsType" xdb:SQLName="LINEITEMS"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="LineItemsType" xdb:SQLType="LINEITEMS_T">
    <xs:sequence>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- The slide screen capture shows the partial annotated purchaseOrder.xsd XML schema file. Before annotating an XML schema, you must first declare the Oracle XML DB namespace. The PurchaseOrder XML schema defines the following two namespaces:
 - `http://www.w3c.org/2001/XMLSchema`. This is reserved by W3C for the Schema for Schemas.
 - `http://xmlns.oracle.com/xdb`. This is the Oracle XML DB namespace which is reserved by Oracle for the Oracle XML DB schema annotations.

The namespace is declared in the XML schema by adding a namespace declaration such as the following to the root element of the XML schema:

`xmlns:xdb="http://xmlns.oracle.com/xdb"`

The use of a namespace prefix, `xdb`, makes it possible to abbreviate the namespace to `xdb` when adding annotations.

Annotated Purchase-Order XML Schema, purchaseOrder.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xdb="http://xmlns.oracle.com/xdb"
    version="1.0">
    <xs:element name="PurchaseOrder" type="PurchaseOrderType" xdb:defaultTable="PURCHASEORDER"/>
    <xs:complexType name="PurchaseOrderType" xdb:SQLType="PURCHASEORDER_T">
        <xs:sequence>
            <xs:element name="Reference" type="ReferenceType" minOccurs="1" xdb:SQLName="REFERENCE"/>
            <xs:element name="Actions" type="ActionsType" xdb:SQLName="ACTIONS"/>
            <xs:element name="Reject" type="RejectionType" minOccurs="0" xdb:SQLName="REJECTION"/>
            <xs:element name="Requestor" type="RequestorType" xdb:SQLName="REQUESTOR"/>
            <xs:element name="User" type="UserType" minOccurs="1" xdb:SQLName="USERID"/>
            <xs:element name="CostCenter" type="CostCenterType" xdb:SQLName="COST_CENTER"/>
            <xs:element name="ShippingInstructions" type="ShippingInstructionsType"
                xdb:SQLName="SHIPPING_INSTRUCTIONS"/>
            <xs:element name="SpecialInstructions" type="SpecialInstructionsType"
                xdb:SQLName="SPECIAL_INSTRUCTIONS"/>
            <xs:element name="LineItems" type="LineItemsType" xdb:SQLName="LINEITEMS"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="LineItemsType" xdb:SQLType="LINEITEMS_T">
        <xs:sequence>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The annotated purchaseOrder schema uses several annotations, including the following:

- **defaultTable annotation in the PurchaseOrder element:** This annotation specifies that XML documents compliant with this XML schema are stored in a database table called purchaseorder.
- **SQLType annotation.**
 - The first occurrence of SQLType specifies that the name of the SQLtype generated from the complexType element PurchaseOrderType is purchaseorder_t.
 - The second occurrence of SQLType specifies that the name of the SQL type generated from the complexType element LineItem Type is lineitem_t, and the SQL type that manages the collection of LineItem elements is lineitem_v.
- **SQLName annotation.** This annotation provides an explicit name for each SQL attribute of purchaseorder_t. For example, the SQLName annotation for element User is USERID.

Annotating an XML Schema by Using DBMS_XMLSHEMA_ANNOTATE

- The DBMS_XMLSHEMA_ANNOTATE PL/SQL package provides subprograms to annotate an XML Schema.
- Each Oracle annotation has specific PL/SQL subprograms.
For example:
 - Use the PL/SQL procedure `setDefaultTable` to add an `xdb:defaultTable` annotation.
 - Use the `removeDefaultTable` to remove an `xdb:defaultTable` annotation.



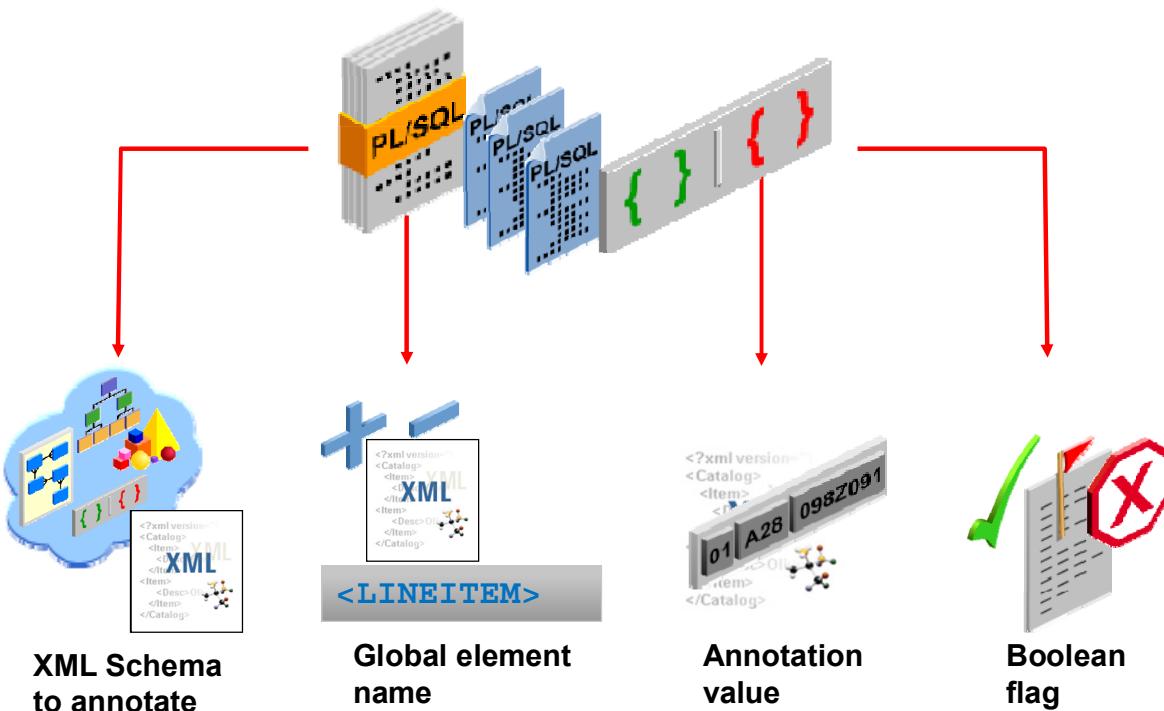
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The DBMS_XMLSHEMA_ANNOTATE PL/SQL package provides subprograms to annotate an XML schema. These subprograms can often be more convenient and less error prone than manually editing the XML schema.

In particular, you can use the PL/SQL subprograms in a script, which you can run at any time or multiple times, as needed. The subprograms can be especially useful if you are using a large XML schema or a standard or other third-party XML schema that you do not want to modify manually.

Each Oracle annotation has specific PL/SQL subprograms. For example, you can use the `setDefaultTable` PL/SQL package procedure to add an `xdb:defaultTable` annotation. You can use the `removeDefaultTable` PL/SQL package procedure to remove an `xdb:defaultTable` annotation.

Annotation Subprogram Parameters



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Each annotation subprogram has the following parameters:

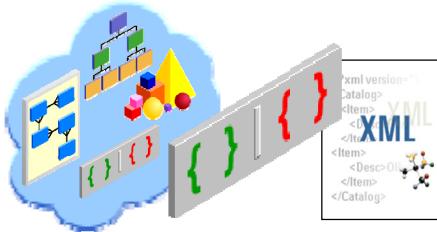
- The XML schema to be annotated. This is an `IN OUT` parameter.
- The name of the global element where the annotation is to be added or removed
- The annotation (XML attribute) value
- A Boolean flag indicating whether any corresponding existing annotation is to be overwritten (by default, it is overwritten).

If the element to be annotated is not a global element, you provide the local element name as an additional parameter. The global and local names together identify the target element. The element with the local name must be a descendent of the element with the global name.

If you use SQL*Plus, you can use the `DBMS_XMLSHEMA_ANNOTATE.printWarnings` PL/SQL package procedure to enable and disable printing of SQL*Plus warnings during the use of other `DBMS_XMLSHEMA_ANNOTATE` subprograms. By default, no warnings are printed. An example of a warning is an inability to annotate the XML schema because there is no element with the name you provided to the annotation subprogram.

Some of the Available Oracle XML DB XML Schema Elements Annotations

Attribute	Description
xdb : defaultTable	Name of the default table generated for each global element when parameter GENTABLES is TRUE. Setting this to the empty string, "", prevents a default table from being generated for the element in question.
xdb : tableProps	TABLE storage clause that is appended to the default CREATE TABLE statement.
xdb : columnProps	COLUMN storage clause that is inserted into the default CREATE TABLE statement.



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This and the next few slides list some of the Oracle XML DB annotations that you can specify in element and attribute declarations. We show only few examples of some of the PL/SQL subprograms in the DBMS_XMLSHEMA_ANNOTATE package that you can use to manipulate the corresponding annotations.

The `xdb : defaultTable` annotation specifies the name of the SQL table into which XML instances of this XML schema are stored. This annotation is useful in cases where the XML data is inserted from APIs and protocols, such as FTP and HTTP(S), where the table name is not specified.

The `xdb : tableProps` annotation specifies the TABLE storage clause that is appended to the default CREATE TABLE statement. This annotation is meaningful mainly for global and out-of-line elements and is applicable to object-relational storage and binary XML storage. The `xdb : columnProps` annotation specifies the COLUMN storage clause that is inserted into the default CREATE TABLE statement. This annotation is useful mainly for elements that get mapped to SQL tables, namely top-level element declarations and out-of-line element declarations.

Some of the Available Oracle XML DB XML Schema Elements Annotations

Attribute	Description
xdb : maintainDOM	If <code>true</code> , instances of this element are stored so that they retain DOM fidelity on output. If <code>false</code> , the output is not guaranteed to have the same DOM action as the input.
xdb : SQLCollType	Name of the SQL collection type that corresponds to this XML element.
xdb : SQLInline	If <code>true</code> , this element is stored inline as an embedded object attribute. If <code>false</code> , a SQL function <code>REF</code> value is stored.
xdb : SQLName	Name of the SQL object attribute that corresponds to each element or attribute defined in the XML schema.
xdb : SQLType	Name of the SQL type corresponding to this XML element declaration.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The `xdb : maintainDOM` annotation is used to determine whether DOM fidelity should be maintained for a given `complexType` definition. If `true`, then instances of this element are stored so that they retain DOM fidelity on output. This implies that all comments, processing instructions, namespace declarations, and so on are retained, in addition to the ordering of elements. If `false`, then the output is not guaranteed to have the same DOM action as the input.

The `xdb : SQLCollType` annotation specifies the name of the SQL collection type that corresponds to this XML element.

The `xdb : SQLInline` annotation: If `true`, then this element is stored inline as an embedded object attribute (or as a collection, if `maxOccurs > 1`). If `false`, then a `REF` value is stored (or a collection of `REF` values, if `maxOccurs > 1`). This attribute is forced to `false` in certain situations, such as cyclic references, where SQL does not support inlining.

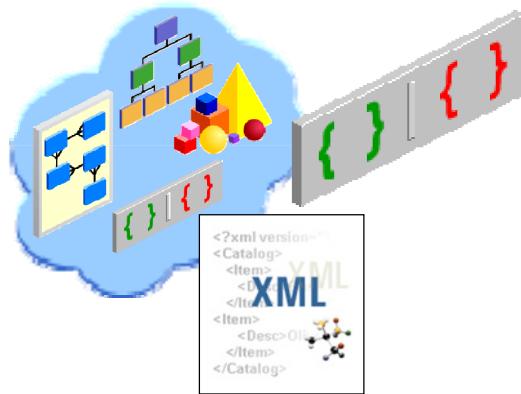
The `xdb : SQLName` annotation specifies the name of the attribute within the SQL object that maps to this XML element.

The `xdb : SQLType` annotation specifies the names of the SQL type corresponding to this XML element declaration.

Note: For detailed descriptions of the available Oracle XML DB XML Schema element annotations and the PL/SQL subprograms in the `DBMS_XMLSCHEMA_ANNOTATE` package, see the *Oracle XML DB Developer's Guide 12c Release 1 (12.1)* documentation guide.

Examples of the Most Commonly Used XML Annotations

- `xdb:defaultTable`
- `xdb:SQLName`
- `xdb:SQLType`
- `xdb:SQLCollType`
- `xdb:maintainDOM`



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You do not need to specify the values for any of the attributes listed in the slide. By default, Oracle XML DB provides appropriate values during the XML Schema registration process. However, if you are using object-relational storage, Oracle recommends that you specify the names of at least the top-level SQL types. Doing so makes it easier for you to reference them later.

DBMS_XMLSHEMA_ANNOTATE Procedure Example: SETDEFAULTTABLE

```
DBMS_XMLSHEMA_ANNOTATE.SETDEFAULTTABLE (
    xmlschema IN OUT XMLTYPE,
    globalElementName IN VARCHAR2,
    tableName IN VARCHAR2,
    overwrite IN BOOLEAN DEFAULT TRUE);
```

Parameter	Description
xmlschema	XML Schema to be annotated
globalElementName	Name of the global element in the schema
tableName	Name being assigned to the table
overwrite	Boolean that indicates whether the procedure overwrites element attributes (The default is TRUE.)



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This procedure sets the name of the table for the specified global element. It is equivalent to using the `xdb:defaultTable="<default_table_name>"` schema annotation for the top-level element.

DBMS_XMLSHEMA_ANNOTATE Procedure Example: setSQLType

```
DBMS_XMLSHEMA_ANNOTATE.SETSQLTYPE (
    xmlschema IN OUT XMLTYPE,
    globalElementName IN VARCHAR2,
    sqlType IN VARCHAR2,
    overwrite IN BOOLEAN DEFAULT TRUE);
```

Parameter	Description
xmlschema	XML Schema to be annotated
globalElementName	Global object (global complex type or global element)
sqlType	SQL type assigned to the named global element
overwrite	Boolean that indicates whether the procedure overwrites element attributes (The default is TRUE.)



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This procedure assigns a SQL type to a global object. There are two overloads. The first overload assigns a SQL type to a global object, such as a global element or global complex type. The second overload assigns a SQL type to a local object.

DBMS_XMLSHEMA_ANNOTATE: Example

```
CREATE TABLE annotation_tab (id NUMBER, inp XMLType, out XMLType);
INSERT INTO annotation_tab VALUES (1, ... unannotated purchaseorder_6
    XML schema...);
DECLARE
schema XMLType;
BEGIN
SELECT t.inp INTO schema FROM annotation_tab t WHERE t.id = 1;
DBMS_XMLSHEMA_ANNOTATE.setDefaultTable(schema, 'PurchaseOrder',
    'PURCHASEORDER');
DBMS_XMLSHEMA_ANNOTATE.setSQLType(schema, 'PurchaseOrderType',
    'PURCHASEORDER_T');
DBMS_XMLSHEMA_ANNOTATE.setSQLName(schema, 'complexType',
    'PurchaseOrderType', 'element', 'Reference', 'REFERENCE');
DBMS_XMLSHEMA_ANNOTATE.setSQLName(schema, 'complexType',
    'PurchaseOrderType', 'element', 'Actions', 'ACTIONS_COLLECTION');
DBMS_XMLSHEMA_ANNOTATE.setSQLName(schema, 'complexType',
    'PurchaseOrderType', 'element', 'User', 'EMAIL');
...
UPDATE annotation_tab t SET t.out = schema WHERE t.id = 1;
END;
/
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The partial slide example uses subprograms in the DBMS_XMLSHEMA_ANNOTATE PL/SQL package to produce the annotated XML Schema.

DBMS_XMLSCHEMA_ANNOTATE: Example

```
-- Registering the annotated XML schema.

DECLARE
schema XMLType;
BEGIN
SELECT t.out INTO schema FROM annotation_tab t WHERE t.id = 1;
DBMS_XMLSCHEMA.registerSchema(
SCHEMABURL =>
  'http://localhost:8080/source/schemas/poSource/xsd/purchase
order_6.xsd',
SCHEMADOC => schema,
LOCAL => TRUE,
GENTYPES => TRUE,
GENTABLES => TRUE);
END;
/
```



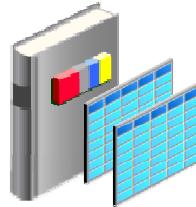
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example registers the annotated XML schema, purchaseorder_6.xsd.

Querying a Registered XML Schema to Obtain Annotations

Catalog View	Description
USER_XML_SCHEMAS	Registered XML Schemas owned by the current user
ALL_XML_SCHEMAS	Registered XML Schemas usable by the current user

```
SELECT SCHEMA
FROM USER_XML_SCHEMAS
WHERE SCHEMA_URL =
'http://localhost:8080/source/schemas/poSource/xsd/purchaseorder_6.xsd';
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The registered version of an XML Schema contains a full set of Oracle XML DB annotations. The annotations are supplied by a user or they are set by default during XML Schema registration.

You can query the `USER_XML_SCHEMAS` and `ALL_XML_SCHEMAS` database views to obtain a registered XML Schema and its annotations, as shown in the slide example. It returns the XML Schema as an `XMLType` instance.

XML Schema Annotation Guidelines for Object-Relational Storage

- Avoid creating unnecessary tables for unused top-level elements.
- Provide your own names for default tables.
- Turn off DOM Fidelity when it is not needed.
- Annotate time-related elements to use time stamp with time zone.
- Add table and column properties.
- Store large collections out of line.



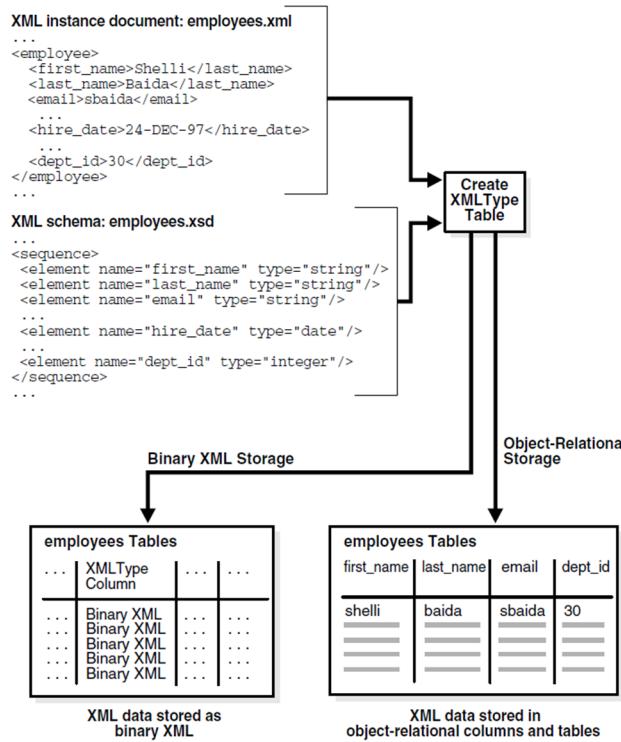
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For XMLType data in object-relational storage, plan carefully so that you optimize performance. Guidelines for XMLType data are similar to those for ordinary relational data, such as entity-relationship model, indexing, data types, and table partitions.

To enable XPath rewrite and achieve optimal performance, you implement many such design choices by using XML Schema annotations. This section provides annotation guidelines to optimize the use of XMLType data in object-relational storage.

The XML Schema annotation guidelines for the object-relational storage model are listed on the slide. For detailed information on the guidelines, see the *Oracle XML DB Developer's Guide 12c Release 1 (12.1)* documentation guide.

How Oracle XML DB Maps XML Schema-Based XMLType Tables



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide diagram shows how Oracle XML DB creates XML schema-based XMLType tables by using an XML document and a mapping specified in an XML schema. Depending on the storage method specified in the XML schema, an XML instance document is stored as a binary XML value in a single XMLType column or in multiple object-relational columns.

Agenda

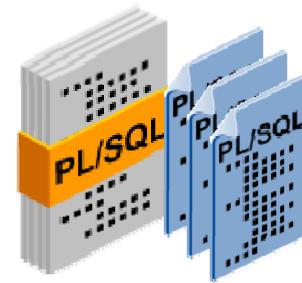
- Oracle XML Schema Annotations
 - Methods for Adding Annotations
 - Reviewing the Purchase-Order XML Schema
 - Annotating an XML Schema by Using the DBMS_XMLSHEMA_ANNOTATE PL/SQL Package
 - Oracle XML DB XML Schema Elements Annotations Examples
- Using the DBMS_XMLSTORAGE_MANAGE PL/SQL Package to Manage and Modify XML Storage.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

DBMS_XMLSTORAGE_MANAGE Package: Overview

- The DBMS_XMLSTORAGE_MANAGE package provides an interface to manage and modify XML storage after schema registration is completed.
- Use subprograms from this package to improve the performance of bulk load operations.



DBMS_XMLSTORAGE_MANAGE
package subprograms

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The DBMS_XMLSTORAGE_MANAGE package contains procedures to manage and modify XML storage after you complete the schema registration. You can use the package's subprograms to improve the performance of bulk load operations. You can disable indexes and constraints before doing a bulk load process, and you can enable them afterwards.

Note: For additional information about the DBMS_XMLSTORAGE_MANAGE package, refer to the Oracle Database PL/SQL Packages and Types Reference 12c Release 1 (12.1) reference guide.

Summary of the DBMS_XMLSTORAGE_MANAGE Package Subprograms

Subprogram	Description
DISABLEINDEXESANDCONSTRAINTS procedure	Disables the indexes and constraints for XMLType tables and XMLType columns
ENABLEINDEXESANDCONSTRAINTS procedure	Rebuilds all indexes and enables the constraints on an XMLType table, including its child tables and out-of-line tables
EXCHANGEPOSTPROC procedure	Enables constraints after exchange partition
EXCHANGEPREPROC procedure	Disables constraints before exchange partition
INDEXXMLREFERENCES procedure	Creates unique indexes on the REF columns of the given XMLType table or the XMLType column of a given table



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The DISABLEINDEXESANDCONSTRAINTS procedure disables the indexes and constraints for XMLType tables and XMLType columns.

The ENABLEINDEXESANDCONSTRAINTS procedure rebuilds all indexes and enables the constraints on an XMLType table, including its child tables and out-of-line tables.

The EXCHANGEPOSTPROC procedure enables constraints after exchange partition.

The EXCHANGEPREPROC procedure disables constraints before exchange partition.

The INDEXXMLREFERENCES procedure creates unique indexes on the REF columns of the given XMLType table or the XMLType column of a given table.

Summary of the DBMS_XMLSTORAGE_MANAGE Package Subprograms

Subprogram	Description
RENAMECOLLECTIONTABLE procedure	Renames a collection table to the given table name
SCOPEXMLREFERENCES procedure	Scopes all XML references (Scoped REF types require less storage space and allow more efficient access than unscoped REF types.)
XPATH2TABCOLMAPPING function	Maps a path expression (in XPath notation or DOT notations) to the corresponding table name and column name



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The RENAMECOLLECTIONTABLE procedure renames a collection table to the given table name.

The SCOPEXMLREFERENCES procedure scopes all XML references. (Scoped REF types require less storage space and allow more efficient access than unscoped REF types.)

The XPATH2TABCOLMAPPING function maps a path expression (in XPath notation or DOT notations) to the corresponding table name and column name.

Quiz

The DBMS_XMLSHEMA_ANNOTATE PL/SQL package provides subprograms to annotate an XML schema. However, editing the XML schema manually is more convenient and less error prone.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

The DBMS_XMLSHEMA_ANNOTATE PL/SQL package provides subprograms to annotate an XML Schema. Using the package's subprograms can be more convenient and less error prone than manually editing the XML Schema.

Summary

In this course, you should have learned how to:

- Identify why and how to annotate XML Schemas
- Review the Purchase-Order, `purchaseOrder.xsd`, XML Schema
- Annotate an XML Schema by using the PL/SQL package `DBMS_XMLSHEMA_ANNOTATE`
- Review some of the Oracle XML DB XML Schema elements annotations
- Identify XML Schema annotation guidelines for object-relational storage
- Review the `DBMS_XMLSTORAGE_MANAGE` PL/SQL package



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 6-1: Overview

This practice covers the following topics:

- Accessing and reviewing an un-annotated XML schema document.
- Creating a new table to hold the XML schema document.
- Inserting the un-annotated XML schema in the newly created table.
- Writing a PL/SQL block to annotate the XML schema.
- Registering the XML schema.
- Verifying the creation of the tables, types, and other objects that were created after the schema registration.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In this practice, you annotate an XML schema by using the PL/SQL package `DBMS_XMLSCHEMA_ANNOTATE`.

Partitioning XMLType Tables and Columns

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

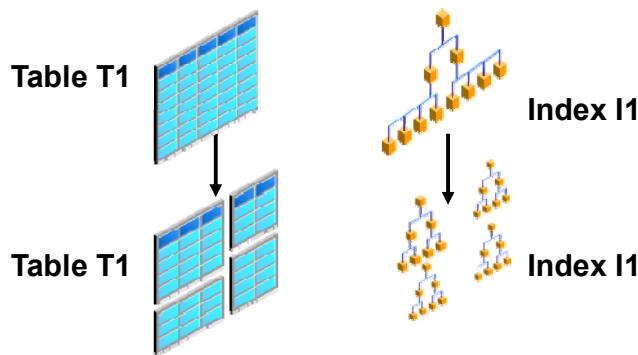
- Identify partitioning concepts
- Conclude the reason for using ordered collection tables (OCTs)
- Partition XMLType tables and columns stored object relationally
- Specify partitioning information for an XMLType base table
- Maintain partitions



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Partitioned Tables and Indexes

- Partitioning is subdividing of large tables and indexes into smaller, more manageable pieces called partitions.
- Partitions can reside in different tablespaces, have distinct storage clauses, and be maintained by granular commands.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Partitioning allows a table or an index to be subdivided into smaller pieces. Each piece of the database object is called a partition. Each partition has its own name, and may optionally have its own storage characteristics. Each partition can be managed individually, and can function independently of other partitions, thus providing a structure that can be better tuned for availability and performance. Each partition or subpartition is a segment and inherits many of the properties that segments have. Different partitions that belong to the same table or index can reside in different tablespaces, have distinct storage clauses, and be maintained by granular commands.

Oracle Database allocates logical space for all the data in the database. The logical units of database space allocation are data blocks, extents, segments, and tablespaces. At a physical level, the data is stored in data files on disk. At the finest level of granularity, Oracle Database stores data in data blocks. One logical *data block* corresponds to a specific number of bytes of physical disk space, for example, 2 KB. An *extent* is a set of logically contiguous data blocks allocated for storing a specific type of information. A *segment* is a set of extents allocated for a specific database object, such as a *table*. For additional information about the logical storage structure, see the *Oracle Database Concepts 12c Release 1 (12.1)* reference guide.

Why Partitioning?

- Data access is faster.
- Maintenance commands can be applied at the partition level, thereby reducing the impact of the scheduled down time for maintenance operations.
- Parallel execution is possible.
- Join operations can be optimized to join “by the partition.”
- Partitions can be load-balanced across physical devices.



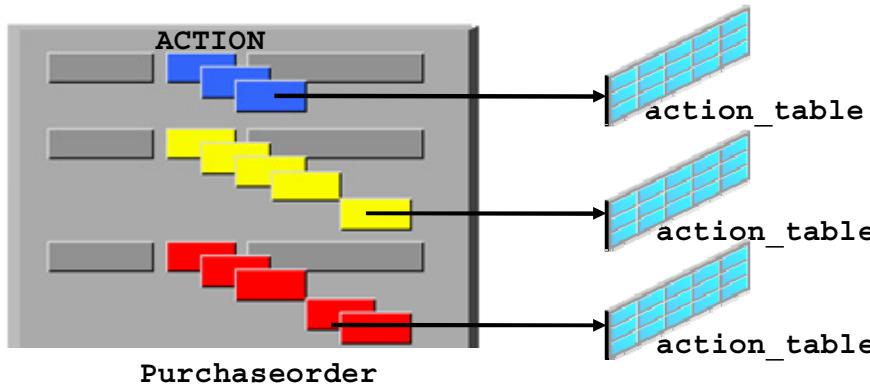
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Partitioning is an important tool for managing large databases. Partitioning allows the DBA to employ a “divide and conquer” methodology for managing database tables, especially as those tables grow. Partitioned tables allow a database to scale for very large data sets while maintaining consistent performance, without unduly impacting the administrative or hardware resources.

Partitioning enables faster data access within an Oracle database. It also enables data management operations such as data loading, index creation and rebuilding, and backup or recovery at the partition level rather than on the entire table. This results in significantly reduced times for these operations. By using partitions, you can reduce the impact of the scheduled down time for maintenance operations. Partitions also provide the advantages of parallel execution.

Ordered Collection Tables

```
CREATE TABLE purchaseorder OF XMLType
  XMLSCHEMA "http://localhost:8080/.../purchaseOrder.xsd"
  ELEMENT "PurchaseOrder"
  VARRAY "XMLDATA"."ACTIONS"."ACTION"
  STORE AS TABLE action_table
  ((PRIMARY KEY (NESTED_TABLE_ID, SYS_NC_ARRAY_INDEX$)))
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

An XML collection is an element that appears multiple times in an `XMLType` table (element that has `maxOccurs > 1`). These collections are mapped into SQL varray values. By default, the entire content of such a varray is stored as a set of rows in an ordered collection table (OCT).

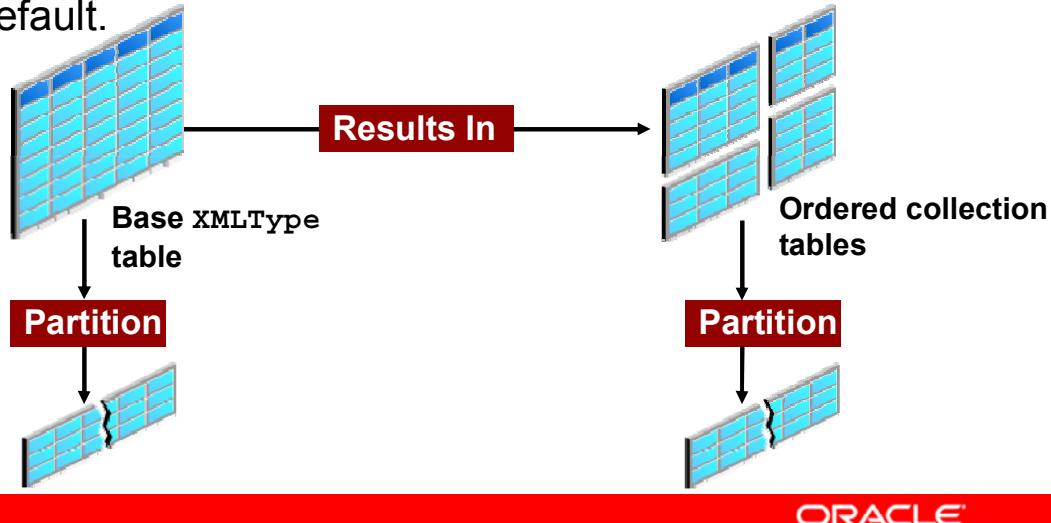
The graphic in the slide demonstrates that each member of the varray that manages the collection of `Action` elements in the `purchaseorder` table is stored in the ordered collection table `action_table`.

When registering an XML schema, if you set `GENTABLES` to `TRUE`, Oracle XML DB creates default tables. Order is preserved among the XML collection elements when they are stored. This results in an ordered collection.

You can store data in an ordered collection as a varray in a table or as a varray in a LOB. Each element in the collection is mapped to a SQL object. The collection of SQL objects is stored as a set of rows in a table, called an ordered collection table (OCT).

Equipartitioning

- Starting with Oracle Database 11g Release 2, Oracle supports equipartitioning of XMLType data stored object relationally.
- When you partition an XMLType table or a table with an XMLType column, any ordered collection tables (OCTs) within the data are automatically partitioned accordingly, by default.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Starting with Oracle Database 11g Release 2, when you partition an XMLType table or a table with an XMLType column using list, range, or hash partitioning, any OCTs within the data are automatically partitioned. The process of the ordered collection tables being partitioned when the base XMLType table is partitioned is known as **equipartitioning**.

When you partition a base table, a corresponding collection-table partition is created for each partition of the base table. The storage attributes for a base table partition are, by default, also used for the corresponding child OCT partitions. A child element is stored in the collection-table partition that corresponds to the base-table partition of its parent element. By default, the name of an OCT partition is the same as its corresponding partition in the base (parent) table.

You can override the storage attributes and name for a given OCT partition.

Advantages of Partitioning an OCT

- Maintaining multiple OCT partitions is easier than maintaining a single large OCT.
- The optimizer eliminates partitions that do not need to be scanned.
- Improved performance: Maintenance commands can be applied at the partition level.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Maintaining multiple ordered collection table partitions is easier than maintaining a single large ordered collection table. For example, you (DBA) store different ordered collection table partitions in different tablespaces. For temporary usage, you can consider a single ordered collection table partition instead of an entire ordered collection table.

Performance of queries that target data stored in ordered collection table partitions is greatly improved. An example of improved query performance is that the ordered collection tables may be pruned during executing of a query. This makes it possible for the user to avoid performing a full scan of an unpartitioned ordered collection table.

Partitioning XMLType Data

You can specify partitioning information for an XMLType base table in two ways:

- During XML schema registration, by using the XML schema annotation `xdb:tableProps`
- During table creation, by using the `CREATE TABLE` command



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In this lesson, you see only an example of partitioning XMLTYPE data during table creation. For information about how to partition XMLTYPE data during XML schema registration, see the chapter titled “XML Schema Storage and Query: Advanced” in the *Oracle XML DB Developer’s Guide 12c Release 1 (12.1)* reference.

Partitioning XMLType Table: During Table Creation

```
CREATE TABLE purchaseorder_7 OF XMLType 1
XMLSCHEMA
  "http://localhost:8080/source/schemas/poSource/xsd/p
   urchaseOrder.xsd"
ELEMENT "PurchaseOrder"
VARRAY "XMLDATA"."LINEITEMS"."LINEITEM" STORE AS TABLE 2
  lineitem_table
((PRIMARY KEY (NESTED_TABLE_ID, SYS_NC_ARRAY_INDEX$)))
PARTITION BY RANGE (XMLDATA.Reference)
(PARTITION p1 VALUES LESS THAN (1000)
VARRAY "XMLDATA"."LINEITEMS"."LINEITEM" STORE AS TABLE 3
  lineitem_p1
(STORAGE (MINEXTENTS 13)),
PARTITION p2 VALUES LESS THAN (2000)
VARRAY "XMLDATA"."LINEITEMS"."LINEITEM" STORE AS TABLE
  lineitem_p2
(STORAGE (MINEXTENTS 13)));
. . .
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

1. PURCHASEORDER: The base XMLType table
2. lineitem_table: The OCT based on purchaseorder.lineitems.lineitem
3. lineitem_p1: The partitioned OCT of LINEITEM_TABLE is equipartitioned with respect to the base XMLType table.

Maintaining a Partition

- By default, maintaining the base table automatically maintains the corresponding OCTs.
- However, the following scenarios require manual maintenance of the OCT partitions:
 - Modifying the default physical storage attributes and the physical storage attributes of a collection partition
 - Moving a collection partition to a different segment, possibly in a different tablespace
 - Renaming a collection partition



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Generally, you perform maintenance only on the base (parent) table. This includes operations such as adding, dropping, and splitting a partition. When you perform partition maintenance on the base table, corresponding maintenance is performed on the OCT table.

Manual Maintaining of Partitions: Examples

```
-- Splits a partition created on the ACCOUNT_XML_LOB2  
-- base XMLType table.
```

```
ALTER TABLE ACCOUNT_XML_LOB2 SPLIT PARTITION p11 AT(50)  
INTO (PARTITION pla, PARTITION plb)  
VARRAY XMLDATA.LINEITEMS.LINEITEM  
INTO (PARTITION line_pl_a, PARTITION line_pl_b)
```

1

```
-- Merges partitions created on the ACCOUNT_XML_LOB2  
-- baseXMLType table.
```

```
ALTER TABLE ACCOUNT_XML_LOB2 MERGE PARTITIONS pla, plb  
INTO PARTITION P1  
VARRAY XMLDATA.LINEITEMS.LINEITEM  
STORE AS TABLE line_pl_a;
```

2

```
-- An exception. Renaming a collection partition is performed  
-- directly on the OCT.
```

```
ALTER TABLE line_tab RENAME LINE_P2 to LINE_P3;
```

3



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Example 1 splits a partition created on the ACCOUNT_XML_LOB2 base XMLType table.

Example 2 merges partitions created on the ACCOUNT_XML_LOB2 base XMLType table.

Example 3 is an exception. Renaming a collection partition is performed directly on the OCT.

Partitioning Binary XML Tables

Starting with Oracle Database 11g Release 2, you can partition binary XML tables, using a virtual column as the partitioning key.

- Unlike real columns, virtual column values are not physically stored on disk but are derived on demand by the evaluation of a function or an expression.
- Virtual columns can be defined within a CREATE or ALTER table operation.

```
CREATE TABLE sales
(sales_amt NUMBER(5),commission AS (sales_amt * 1.15) );
```

- Virtual columns can be indexed, and used in queries, DML and DDL statements like other table column types.
- Tables and indexes can be partitioned on a virtual column and even statistics can be gathered upon them.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle Database 11g Release 1 supported the creation of a binary XML table with virtual columns. Starting with Oracle Database 11g Release 2, you can partition binary XML tables using virtual columns.

Virtual columns are those whose values are derived by computation of a function or an expression. These columns can be specified during a CREATE or ALTER table operation.

Virtual columns share the same SQL namespace as other real table columns and conform to the data type of the underlying expression that describes it. These columns can be used in queries like any other table columns, thereby providing a simple, elegant, and consistent mechanism of accessing expressions in a SQL statement.

The values for virtual columns are not physically stored in the table row on disk; rather they are evaluated on demand. Virtual columns can be used like any other table columns. They can be indexed, and used in queries, data manipulation language (DML) and data definition language (DDL) statements. Tables and indexes can be partitioned on a virtual column and even statistics can be gathered upon them. You can use virtual column partitioning to partition key columns defined on the virtual columns of a table.

Frequently, business requirements to logically partition objects do not match the existing columns in a one-to-one manner. A partitioning strategy defined on virtual columns enables a more comprehensive match of the business requirements.

Using Virtual Columns to Partition Binary XML Tables

- You can create a virtual column based on an XML element or attribute by defining it in terms of a SQL expression that involves that element or attribute.
- The virtual columns that you create on an `XMLType` table are hidden and do not show up in a `DESCRIBE` statement.
- The data type of virtual columns must be constant.
- You can use range, hash, and list partitioning.
- Use the `ALTER TABLE` command to change partitions.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To partition XML data according to the values of individual elements or attributes, you cannot use the standard approach that you use for relational data. This is because individual XML elements and attributes are not mapped to individual database columns or tables. Therefore, you create and use virtual columns. This approach applies only to the XML data that is stored as binary XML.

You can create a virtual column on `XMLType` data as you would create using any other type of data, but by using a slightly different syntax. However, you cannot specify any constraints in association with the column definition. You can create a virtual column based on an XML element or attribute by defining it in terms of a SQL expression that involves that element or attribute. This means that you create a function-based column.

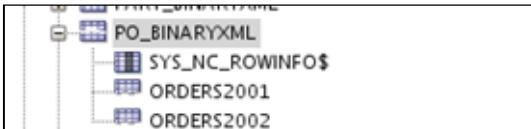
The virtual columns that you create on an `XMLType` table are hidden and do not show up when you use the `DESCRIBE` statement. If you create virtual columns on a table that has an `XMLType` column, you see the virtual columns along with the nonvirtual columns listed by a `DESCRIBE` operation.

To use a virtual column for partitioning, its data type must be constant. If the `XMLType` data in the column or table is mixed, you must cast the functional expression to ensure that the same data type is used for all the rows in the virtual column.

Steps to Partition an XMLTYPE Table Stored as Binary XML

- Define virtual columns for the desired XML data.
- Use virtual columns to partition the XMLType data as a whole.

```
CREATE TABLE po_binaryxml OF XMLType
XMLTYPE STORE AS BINARY XML
VIRTUAL COLUMNS
(DATE_COL AS (XMLCast(XMLQuery('/PurchaseOrder/@orderDate'
PASSING OBJECT_VALUE RETURNING CONTENT)
AS DATE)))
PARTITION BY RANGE (DATE_COL)
(PARTITION orders2001 VALUES LESS THAN (to_date('01-JAN-
2002')) ,
PARTITION orders2002 VALUES LESS THAN (MAXVALUE)) ;
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide partitions an XMLType table that is stored as binary XML. The example first uses the XMLCast and XMLQuery SQL functions to create a function-based virtual column, DATE_COL, which targets the orderDate element in a purchase order document. The example then uses DATE_COL to partition the table as a whole.

Note

- For best performance, use as the partitioning key an XPath expression that occurs within the first 32 KB of the text form of the XML document.
- To partition an XMLType table, you must use a virtual column. For a table with an XMLType column, Oracle recommends you partition on the relational column.

Partitioning Data Dictionary Views

- **USER_TAB_PARTITIONS:**
 - Describes partition-level partitioning information, partition storage parameters, and partition statistics generated by the DBMS_STATS package for all partitions owned by the current user.
 - Its columns are the same as those in ALL_TAB_PARTITIONS.
- **USER_NESTED_TABLES**
 - Describes the nested tables in tables owned by the current user.
 - Its columns are the same as those in ALL_NESTED_TABLES.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLIndex Partitioning and Parallelism

- If you partition an XMLType table, or a table with an XMLType column, using range, list, or hash partitioning, you can also create an XMLIndex index on the table.
- You must use the keyword LOCAL when you create the XMLIndex index. The index and all of its storage tables are locally equipartitioned with respect to the base table.
- You cannot create an XMLIndex index on a partitioned table if you do not use the keyword LOCAL.
- You cannot create an XMLIndex with hash partitioning.
- You can use a PARALLEL clause when you create or alter an XMLIndex index.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If you partition an XMLType table or a table with an XMLType column using range or list partitioning, you can also create an XMLIndex index on the table. If you use the keyword LOCAL when you create the XMLIndex index, the index and all of its storage tables are locally equipartitioned with respect to the base table. If you do not use the keyword LOCAL, you cannot create an XMLIndex index on a partitioned table. Also, if you hash-partition a table, you cannot create an XMLIndex index on it.

You can use a PARALLEL clause (with optional degree) when creating or altering an XMLIndex index to ensure that index creation and maintenance are carried out in parallel. If the base table is partitioned or enabled for parallelism, this can improve performance for both DML operations (INSERT, UPDATE, and DELETE) and index DDL operations (CREATE, ALTER, and REBUILD).

Specifying parallelism for an index can consume more storage, because storage parameters apply separately to each query server process. For example, an index created with an INITIAL value of 5 MB and a parallelism degree of 12 consumes at least 60 MB of storage during index creation.

Quiz

Why is partitioning an important tool for managing large databases?

- a. It provides faster data access.
- b. It reduces the impact of the scheduled down time for maintenance operations because maintenance commands can be applied at the partition level.
- c. Partition allows parallel execution.
- d. Join operations can be optimized to join “by the partition.”
- e. All of the above



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: e

Quiz

Equipartitioning is the process of:

- a. Dividing all database tables equally
- b. The ordered collection tables being partitioned when the base XMLType table is partitioned
- c. Dividing XMLType database tables equally



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

Quiz

By default, maintaining the base table automatically maintains the corresponding OCTs.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a

Quiz

Oracle Database supports partitioning binary XML tables, by using only real columns as the partitioning key.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

Summary

In this lesson, you should have learned how to:

- Identify partitioning concepts
- Conclude the reason for using ordered collection tables
- Partition XMLType tables and columns stored object relationally
- Specify partitioning information for an XMLType base table
- Maintain partitions



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 7: Overview

This practice covers partitioning XML data.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Using XQuery to Retrieve XML Data in Oracle XML DB



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Explain the different types of XML queries
- Describe XQuery support in Oracle XML DB
- Query a relational table as if it were XML data
- Use SQL/XML standard functions and XQuery to retrieve XML data that is stored in `XMLType` tables
- Query an XML document in the Oracle XML DB Repository
- Use a namespace with XQuery
- Use the XQuery `XQUERY` command in `SQL*Plus`



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In this lesson, you learn:

- To retrieve XML data
- How XQuery is supported in Oracle XML DB
- To query a relational table as if it were XML data
- To query native `XMLType` data

This lesson focuses on retrieving XML data by using the following SQL/XML functions:

- `XMLQuery`
- `XMLTable`
- `XMLExists`
- `XMLCast`

You also learn how to use `fn:collection` in association with the `XMLTable` and `XMLQuery` functions to extract data that is stored in relational tables and to extract native `XMLType` data.

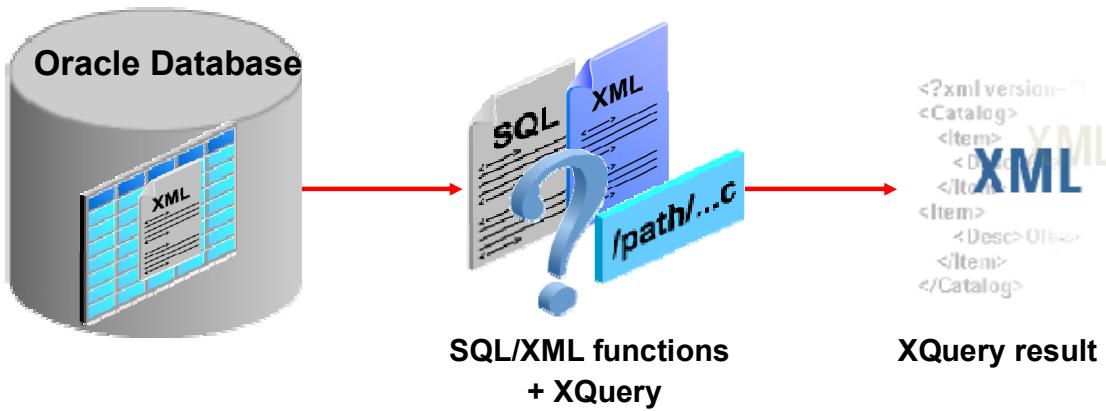
Lesson Agenda

- Retrieving XML content
- Using FLWOR expressions: Review
- XQuery support in Oracle Database
- Querying the database: Relational data
- Querying the database: XMLType data
- Querying XMLType data by using SQL/XML standard functions
- Querying an XML document in the Oracle XML DB Repository
- Using a namespace with XQuery
- XQuery support in SQL*Plus: the XQUERY command



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Retrieving XML Content: Overview

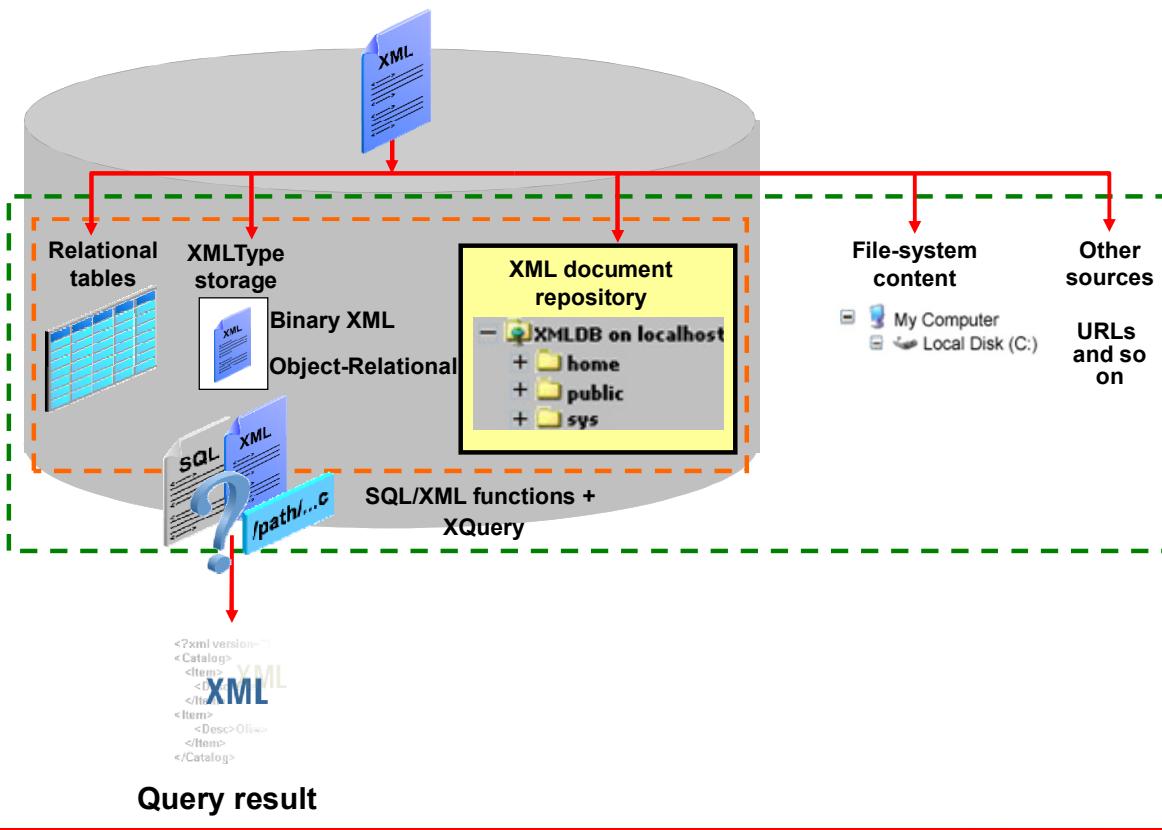


ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use SQL/XML functions along with their XQuery string parameters to retrieve XML content from an Oracle Database.

Types of XML Queries



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Based on the storage of XML data, XML queries can be categorized as follows:

- **Table-based queries:** XML data stored in Oracle Database tables are retrieved by using table-based queries. To perform these queries, you can use XQuery wrapped in SQL/XML functions. Table-based queries can be further classified into the following types:
 - **Relational-data query:** Generates XML content from relational tables
 - **XMLType-data query:** Retrieves XML content stored in XMLType table or XMLType columns
- **Non-table-based queries:**
 - **Oracle XML DB Repository:** In the lesson titled “Working With the Oracle XML DB Repository,” you learn about SQL access of XML files stored in the Oracle XML DB Repository folders by using `PATH_VIEW` and `RESOURCE_VIEW`. You can also use XQuery for querying an XML file stored in the Oracle XML DB Repository by using the `fn:doc()` and `fn:collection()` functions.

External data sources: XQuery supports retrieval of XML data that is stored externally in locations such as file systems, URLs, and so on.

You can use XQuery to perform both table-based and non-table-based queries. But you cannot use SQL/XML functions to query external data sources.

This lesson covers table-based queries. Non-table-based queries are covered in the lesson titled “Using XQuery with Oracle XML DB: Additional Use Cases.”

XQuery: Review

- XQuery is the W3C standard language designed for querying (finding and extracting elements and attributes).
- XQuery is to XML what SQL is to database tables.
- ***XQuery is built on XPath expressions.***
- By using XQuery you can query both structured and unstructured data:
 - Relational databases
 - XML documents
 - Other data sources with XML data view



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XQuery is the W3C standard language designed for querying, transforming, and accessing XML and relational data. XQuery is similar to SQL in many ways. Like SQL is designed for querying structured, relational data, XQuery is designed especially for querying unstructured, XML data from a variety of data sources.

You can use XQuery to query XML data wherever it is found, whether it is stored in the database tables, or available through Web services. In addition to querying XML data, you can use XQuery to construct XML data.

Oracle XML DB supports a native XQuery compilation engine that can parse and compile XQuery expressions into SQL-native structures for evaluation. This native execution significantly improves the performance of XQuery expressions in Oracle Database.

XQuery Review: books.xml Document Example

```
<?xml version="1.0" encoding="UTF-8"?>
<bookshop>
  <book category="CHILDREN">
    <title language="en">Love You Forever</title>
    <author>Robert Munsch</author>
    <year>1995</year>
    <price>4.98</price>
  </book>
  <book category="COMPUTERS">
    <title language="en">XML: Visual QuickStart Guide</title>
    <author>Kevin Howard Goldberg</author>
    <year>2008</year>
    <price>23.17</price>
  </book>
  <book category="COMPUTERS">
    <title language="en">Beginning XML, 5th Edition</title>
    <author>Joe Fawcett</author>
    <author>Liam R.E. Quin</author>
    <author>Danny Ayers</author>
    <year>2012</year>
    <price>25.97</price>
  </book>
</bookshop>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XML document in the slide will be used in the example on the next page.

FLWOR Expressions: Review

A FLWOR expression is one of the most important and powerful expressions in XQuery syntax. It supports iteration and binding of variables. It stands for:

- for
- let
- where
- order by
- return

Use the `doc()` function to open the `books.xml` file

```
for $i in doc("books.xml")/bookshop/book
where $i/price > 20
order by $i/title
return $i/title
```

```
<title language="en">Beginning XML, 5th Edition</title>
<title language="en">XML: Visual QuickStart Guide</title>
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Expressions are the basic building blocks of XQuery. They are case-sensitive.

The acronym FLWOR (pronounced “flower”) represents the XQuery clauses `for`, `let`, `where`, `order by`, and `return`. A FLWOR expression has at least one `for` or `let` clause and a `return` clause; single `where` and `order by` clauses are optional.

- `for`: Analogous to the `FROM` clause of a SQL `SELECT` query. By using the `for` clause you can iterate across a range of sequence values and bind each one of them to one or more bind variables. At each iteration, the variables are bound in the order in which they appear.
- `let`: Analogous to the SQL `SET` statement. You can use the `let` clause to define variables and assign them, in turn, during iteration through a `for` clause. You can bind one or more variables. Just as with `FOR`, a variable can be bound by `let` to a value that is computed by using another variable that is listed previously in the binding list of `let` (or an enclosing `for` or `let`).
- `where`: Filters the `for` and `let` variable bindings according to a condition. This is similar to the SQL `WHERE` clause.
- `order by`: Arranges the result (returned by the `WHERE` clause) in ascending or descending order

return: Constructs a result from the ordered, filtered values. This is the result of the FLWOR expression as a whole, which can be compared to a SELECT statement.

FLWOR expression syntax:

```
for $identifier in expr(expr)
let $identifier := expr(expr)
where conditional_expr
order by identifier("ascending" | "descending")
return expr_containing_result
```

The example on the previous slide displays a FLWOR expression to select the title elements under the bookshop and book elements in the books.xml file where the book price value is greater than 20.

Explanation of the FLWOR expression clauses:

- The **for** clause assigns all book elements under the bookshop element in the books.xml file to a variable called \$i.
- The **where** clause is used to filter \$i so that only book elements with a price element with a value greater than 20 are selected.
- The **order by** clause defines the sort order which is ascending by default. The **order by** clause will sort on the title element.
- The **return** clause returns the text value of the resulting title elements.

Note

- XQuery expressions are of the following types: Primary, Sequence, FLWOR, Path, Arithmetic, Logical, Conditional, and Quantified.
- XQuery includes built-in functions for string values, numeric values, sequence manipulation, Boolean values, and so on.

Examples:

- fn:abs(): Returns the absolute value of the argument
- fn:ceiling(): Returns the smallest number with no fractional part that is greater than or equal to the argument
- fn:count(): Returns the number of items in a sequence
- fn:avg(): Returns the average of a sequence of values
- fn:max(): Returns the maximum value from a sequence of comparable values
- fn:min(): Returns the minimum value from a sequence of comparable values
- fn:sum(): Returns the sum of a sequence of values
- fn:concat(): Accepts two or more arguments and casts them to xs:string, and then returns the xs:string that is the concatenation of the values of its arguments after conversion

You learned about the XQuery expressions and functions in the course titled *Oracle 11g: XML Fundamentals*.

Static Type-Checking of XQuery Expressions

```
connect oe/oe;
SELECT xtab.poref, xtab.usr, xtab.requestor
FROM purchaseorder p,
XMLTable(
'for $i in /PurchaseOrder
where $i/costcenter eq "A10"
return $i'
PASSING value(p)
COLUMNS
poref VARCHAR2(20) PATH 'Reference',
usr VARCHAR2(20) PATH 'User'
DEFAULT 'Unknown',
requestor VARCHAR2(20) PATH 'Requestor') xtab;
```

```
Error at Command Line:3 Column:5
Error report:
SQL Error: ORA-19276: XPath step specifies an invalid element/attribute name:
19276. 00000 -  "XP0005 - XPath step specifies an invalid element/attribute name: (%$)"
*Cause:   The XPath step specified invalid element or attribute name that did not match
*Action:  Correct the element or attribute name as the name may be mis-spelled.
Connection created by CONNECT script command disconnected
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle XML DB supports static (compile-time) type-checking for XQuery expressions. The example in the slide returns an error because the CostCenter node is specified in the wrong case. XQuery static type-checking finds a mismatch between an XPath expression and its target XML schema-based data. This demonstrates that XML expressions are case-sensitive. The following is the correct syntax for the slide example:

```
SELECT xtab.poref, xtab.usr, xtab.requestor
FROM purchaseorder p,
XMLTable('for $i in /PurchaseOrder
where $i/CostCenter eq "A10" return $i'
PASSING value(p)
COLUMNS
poref VARCHAR2(20) PATH 'Reference',
usr VARCHAR2(20) PATH 'User'
DEFAULT 'Unknown',
requestor VARCHAR2(20) PATH 'Requestor') xtab;
```

Note: The PASSING and COLUMNS clauses are explained later in this lesson.

XPath Expressions Verification

```

SELECT  *
FROM XMLTABLE (
  for $i in fn:collection("oradb:/HR/DEPARTMENTS") /ROW
  return
    <Department id ="{$i/DEPARTMENT_ID}">
      <Employees>
        {for $j in fn:collection("oradb:/HR/EMPLOYEES") /ROW
          where $j/DEPARTMENT_ID eq $i/DEPARTMENT_ID
          return ($j/FIRST_NAME,
                  $j/LAST_NAME,
                  $j/SALARY) }
      </Employees>
    </Department>'  );
  
```

COLUMN_VALUE

```

<Department id="10"><Employees><FIRST_NAME>Jennifer</FIRST_NAME><LAST_NAME>Whale
<Department id="20"><Employees><FIRST_NAME>Michael</FIRST_NAME><LAST_NAME>Hartst
<Department id="30"><Employees><FIRST_NAME>Den</FIRST_NAME><LAST_NAME>Raphaely</

```

```

<Department id="260"><Employees></Employees></Department>
<Department id="270"><Employees></Employees></Department>

```

27 rows selected.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XPath expressions are verified as shown in the slide example. The following example is similar to the code example in the slide; however, it returns an error because the `ROW` expression is missing in the syntax.

```

SELECT  *  FROM XMLTABLE (
  for $i in fn:collection("oradb:/HR/DEPARTMENTS")
  return
    <Department id ="{$i/DEPARTMENT_ID}">
      <Employees>
        {for $j in fn:collection("oradb:/HR/EMPLOYEES")
          where $j/DEPARTMENT_ID eq $i/DEPARTMENT_ID
          return ($j/FIRST_NAME,
                  $j/LAST_NAME,
                  $j/SALARY) }
      </Employees>
    </Department>'  );
  
```

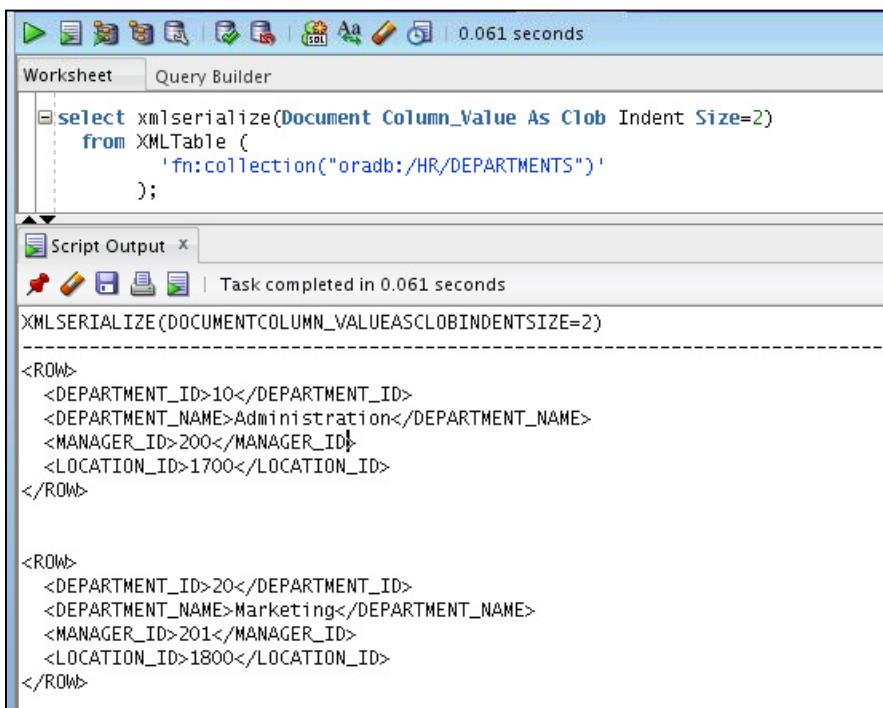
Error at Command Line:2 Column:1
 Error report:
 SQL Error: ORA-19276: XPath step specifies an invalid element/attribute name: (DEPARTMENT_ID)
 19276. 00000 - "XP0005 - XPath step specifies an invalid element/attribute name: (%)"
 *Cause: The XPath step specified invalid element or attribute name that did not match any nodes acco
 *Action: Correct the element or attribute name as the name may be mis-spelled.

Oracle XML DB allows XQuery to operate directly on relational data. Two distinct techniques can be used for this, one is XML-centric, and the other is SQL-centric. Both techniques are based on the concept of creating an XML representation of the data in relational tables. There is no difference in terms of efficiency; the decision about which approach to adopt is a matter of personal preference.

- The first technique uses a canonical mapping to generate an XML representation of each row in a relational table. Non-canonical XML representations of the relational data can then be created by using XQuery to transform from the canonical representation into the required format.
- The second approach involves a SQL-centric approach, based on the SQL/XML publishing functions. The SQL/XML publishing functions allow a SQL statement to return one or more XML documents rather than a traditional tabular result set.

Both techniques can be used to create XML views that allow XQuery-based operations to be performed on relational data.

The following example shows how to use the canonical mapping technique to generate an XML representation of each row in the `departments` table in the `HR` schema. The partial output shows only 2 rows from the 27 rows returned.



A screenshot of the Oracle SQL Developer interface. The top menu bar includes 'File', 'Edit', 'Tools', 'Help', and a 'SQL' icon. The status bar at the bottom says '0.061 seconds'. The main window has two tabs: 'Worksheet' and 'Query Builder'. The 'Worksheet' tab is active, showing the following XQuery code:

```
select xmlserialize(Document Column_Value As Clob Indent Size=2)
  from XMLTable (
    'fn:collection("oradb:/HR/DEPARTMENTS")'
);
```

Below the code, the 'Script Output' tab is visible, showing the results of the query:

```
XMLSERIALIZE(DOCUMENTCOLUMN_VALUEASCLOBINDENTSIZE=2)

<ROW>
<DEPARTMENT_ID>10</DEPARTMENT_ID>
<DEPARTMENT_NAME>Administration</DEPARTMENT_NAME>
<MANAGER_ID>200</MANAGER_ID>
<LOCATION_ID>1700</LOCATION_ID>
</ROW>

<ROW>
<DEPARTMENT_ID>20</DEPARTMENT_ID>
<DEPARTMENT_NAME>Marketing</DEPARTMENT_NAME>
<MANAGER_ID>201</MANAGER_ID>
<LOCATION_ID>1800</LOCATION_ID>
</ROW>
```

Each document has a root node called `ROW`. The root node contains one element for each column in the table. The element names will be generated from the column names.

On the previous slide, the query shows how XQuery can be used to transform the canonical mapping into a more useful XML format.

Note: Oracle canonical XML format is also covered in lesson 12 titled Generating XML Data and in Practice 18-14, XML Access to Relational Content in the case study.

XQuery Support in Oracle Database



SQL/XML functions:

- XMLQuery
- XMLTable
- XMLCast
- XMLExists



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle XML DB supports the XQuery language through a native implementation of the SQL/XML functions, XMLQuery and XMLTable. Oracle XML DB supports the SQL*Plus command XQUERY, which allows you to enter XQuery expressions directly. In effect, the XQUERY command turns SQL*Plus into an XQuery command-line interpreter.

Oracle XML DB compiles XQuery expressions that are passed as arguments to the SQL functions XMLQuery, XMLTable, XMLExists, and XMLCast. This compilation produces SQL query blocks and operator trees that use SQL/XML functions and XPath functions. A SQL statement that includes XMLQuery, XMLTable, XMLExists, or XMLCast is compiled and optimized as a whole, leveraging both relational database and XQuery-specific optimization technologies.

You can use XQuery functions `fn:doc` and `fn:collection` to query resources in Oracle XML DB Repository. In this section, the XQuery function `fn:collection` is used to query data in database tables and views. Later in the lesson, the XQuery function `fn:collection` will be used to query resources in Oracle XML DB Repository.

XQuery Support in Oracle Database

Function	Description
XMLQuery	Takes an XQuery expression as a string literal, an optional context item, and other bind variables and returns the result of evaluating the XQuery expression with these input values. It has two options, RETURNING CONTENT or RETURNING SEQUENCE . <i>XMLQuery is typically used in a SELECT list.</i>
XMLTable	Maps the result of an XQuery evaluation into relational rows and columns so that the user can query the XQuery result as a virtual relational table by using SQL. <i>XMLTable can be only used in the FROM clause.</i>
XMLExists	Checks whether a given XQuery expression returns a nonempty XQuery sequence. If the result of evaluating the XQuery expression is a nonempty XQuery sequence, the function returns TRUE; otherwise, it returns FALSE. <i>XMLExists() is typically used in a SQL WHERE clause.</i>
XMLCast	Allows you to cast items from XML to SQL scalar data type. <i>XMLCast is typically used in a SELECT list.</i>

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLQuery: XMLQuery takes an XQuery expression as a string literal, an optional context item, and other bind variables and returns the result of evaluating the XQuery expression with these input values. It has two options, RETURNING CONTENT or RETURNING SEQUENCE, to indicate whether the result from the XQuery evaluation should be converted to an XML (CONTENT) type or returned as an XML (SEQUENCE) type.

XMLTable: XMLTable is used to map the result of an XQuery evaluation into relational rows and columns so that the user can query the XQuery result as a virtual relational table by using SQL. XMLTable can be used only in the FROM clause of SQL queries.

XMLExists: The XMLExists function introduced in Oracle Database 11g checks whether a given XQuery expression returns a nonempty XQuery sequence. If the result of evaluating the XQuery expression is a nonempty XQuery sequence, the function returns TRUE; otherwise, it returns FALSE. You use XMLExists() in a SQL WHERE clause.

XMLCast: The XMLCast function was introduced in Oracle Database 11g. This function allows you to cast items from XML to SQL scalar data type.

Both `XMLQuery` and `XMLTable` evaluate an XQuery expression. In the XQuery language, an expression always returns a sequence of items. `XMLQuery` aggregates the items in this sequence to return a single XML document or fragment. `XMLTable` returns a SQL table whose rows each contain one item from the XQuery sequence.

Note: SQL/XML functions can be categorized into two types:

- Functions that you can use to query and access XML content. The `XMLQuery`, `XMLTable`, and `XMLExists` functions belong to this category.
- Functions that you can use to generate XML data from the result of a SQL query. These include the `XMLElement`, `XMLAttributes`, `XMLAgg`, and `XMLForest` functions.

Oracle XML DB adds some XQuery functions to those provided in the World Wide Web Consortium (W3C) standard. These additional functions are in the Oracle XML DB namespace, <http://xmlns.oracle.com/xdb>, which uses the predefined prefix `ora`.

To query a relational table or view as if it were XML data, without having to first create a SQL/XML view on top of it, use XQuery function `fn:collection` within an XQuery expression, passing as argument a URI that uses the URI-scheme name `oradb` together with the database location of the data.

XQuery Full Text Search Using Contains Text: In the lesson titled “Search XML Content Using XQuery Full-Text” in this course, you learn how to use XQuery Full Text search by using the `contains text` expression.

For more information about `contains text` expression, see *Oracle XML DB Developer’s Guide, 12c Release 1 (12.1)*.

XMLQuery

Syntax:

```
XMLQuery(<XQuery-string-literal>
          [PASSING [by value] <expression-
returning-XMLType>]
          RETURNING CONTENT)
```

Example:

```
SELECT XMLQUERY(' (1,2+3, "a") '
RETURNING CONTENT) as output
FROM DUAL;
```

```
Connected
OUTPUT
```

```
-----  
1 5 a
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLQuery function is very flexible. You can provide numbers, strings, and XML nodes as arguments to this function. In the slide example, the argument is an XQuery sequence of two numerical expressions and a string. In the syntax, the XQuery string literal is a complete XQuery expression, including the prolog and so on. A prolog is a series of declarations and imports that define the processing environment for the fragment of XQuery code. The PASSING clause contains SQL expressions, each returning an XMLType that is used as the context for evaluating the XQuery expression. Oracle XML DB supports only passing “by value” and not passing “by reference,” so the BY VALUE clause is implicit and can be omitted.

```
SELECT XMLQuery('<TEST>34</TEST>'
RETURNING CONTENT) AS xmlres
FROM DUAL;
```

```
XMLRES
```

```
-----  
<TEST>34</TEST>
```

In the preceding example, the XQuery string <TEST>34</TEST>, which is an element constructor, is invoked through the XMLQuery function. Because this XQuery string does not require a context item or bind variables, you do not have to pass additional parameters. XQuery returns an XQuery sequence consisting of a single element node. returning content is supplied to force the XQuery sequence into the XML (CONTENT) type.

URI Scheme oradb: Querying Table or View Data with the XQuery fn:collection Function

- You can use the XQuery functions `fn:doc` and `fn:collection` to either query resources in Oracle XML DB Repository or query data in database tables and views.
- To do this, you pass the `fn:collection` function a URI argument that specifies the table or view to query.
- The Oracle URI scheme `oradb` identifies this usage. Without it, the argument is treated as a repository location.
- The table or view that is queried can be relational or of type `XMLType`.
- The `fn:collection` function replaces the `ora:view` function which was deprecated in Oracle Database 11g Release 2.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use XQuery functions `fn:doc` and `fn:collection` to query resources in Oracle XML DB Repository. In this lesson, the XQuery function `fn:collection` is used to query data in database tables and views. To do this, you pass function `fn:collection` a URI argument that specifies the table or view to query. The Oracle URI scheme `oradb` identifies this usage: without it, the argument is treated as a repository location.

The table or view that is queried can be relational or of type `XMLType`. If relational, its data is converted on the fly and treated as XML. The result returned by `fn:collection` is always an XQuery sequence.

- For an `XMLType` table, the root element of each XML document returned by `fn:collection` is the same as the root element of an XML document in the table.
- For a relational table, the root element of each XML document returned by `fn:collection` is `ROW`. The children of the `ROW` element are elements with the same names (uppercase) as columns of the table. The content of a child element corresponds to the column data. That content is an XML element if the column is of type `XMLType`; otherwise (the column is a scalar type), the content is of type `xs:string`.

URI Scheme oradb: Querying Table or View Data with the XQuery fn:collection Function

The format of the URI argument passed to fn:collection is as follows:

- For an XMLType or relational table or view, TABLE, in database schema DB-SCHEMA:
 - oradb:/DB-SCHEMA/TABLE/
- For an XMLType column in a relational table or view:
 - oradb:/DB-SCHEMA/REL-TABLE/ROWPRED/X-COL
- Example:

```
...
'fn:collection("oradb:/OE/WAREHOUSES/ROW [WAREHOUSE_ID
< 6 ] /WAREHOUSE_SPEC")'
...
'
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The format of the URI argument passed to fn:collection is as follows:

- For an XMLType or relational table or view, TABLE, in database schema DB-SCHEMA:
oradb:/DB-SCHEMA/TABLE/
You can use PUBLIC for DB-SCHEMA if TABLE is a public synonym or TABLE is a table or view that is accessible to the database user currently logged in.
- For an XMLType column in a relational table or view:
oradb:/DB-SCHEMA/REL-TABLE/ROWPRED/X-COL
 - REL-TABLE is a relational table or view
 - PRED is an XPath predicate that does not involve any XMLType columns
 - X-COL is an XMLType column in REL-TABLE
 - PRED is optional
 - DB-SCHEMA, REL-TABLE, and X-COL are required

The XQuery expression example in the slide represents all XML documents in XMLType column warehouse_spec of table oe.warehouses, for the rows where column warehouse_id has a value of less than 6.

fn:collection Function: Example

```
-- Run code_08_21_n.sql to create the table first. Run this
-- script in SQL*Plus.

SELECT
XMLQUERY('fn:collection("oradb:/HR/EMP_DEPARTMENTS")'
RETURNING CONTENT)
FROM dual;
```

```
SQL> set long 20000
SQL> set pagesize 100
SQL> /
XMLQUERY('FN:COLLECTION("ORADB:/HR/EMP_DEPARTMENTS")' RETURNINGCONTENT)
-----  
<?xml version="1.0" encoding="US-ASCII"?>
<departments>
<department num="1">
<department_id>10</department_id>
<department_name>Administration</department_name>
</department>
<department num="2">
<department_id>20</department_id>
<department_name>Marketing</department_name>
</department>
<department num="3">
<department_id>30</department_id>
<department_name>Purchasing</department_name>
</department>
<department num="4">
<department_id>40</department_id>
<department_name>Human Resources</department_name>
</department>
</departments>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows the use of fn:collection to query the EMP_DEPARTMENTS table. You need to create this table before running the slide code example.

The EMP_DEPARTMENTS table is created by using the following commands:

```
CREATE TABLE EMP_DEPARTMENTS OF XMLTYPE;
INSERT INTO EMP_DEPARTMENTS VALUES (XMLType(bfilename('XML_DIR',
'department.xml'),nls_charset_id('AL32UTF8')));
```

```
Connected
table EMP_DEPARTMENTS created.
1 rows inserted.
Connection created by CONNECT script command disconnected
```

XMLTable

XMLTable:

- Is used to map the results of an XQuery evaluation into relational rows and columns
- Can be used only in the SQL FROM clause

Syntax:

```
XMLTABLE ( [XMLNAMESPACES <string>]
<XQuery-string-literal>
[BY [PASSING [BY VALUE] <expression-returning-XMLType>]
[COLUMNS [FOR ORDINALITY | datatype ]
RETURNING CONTENT)
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

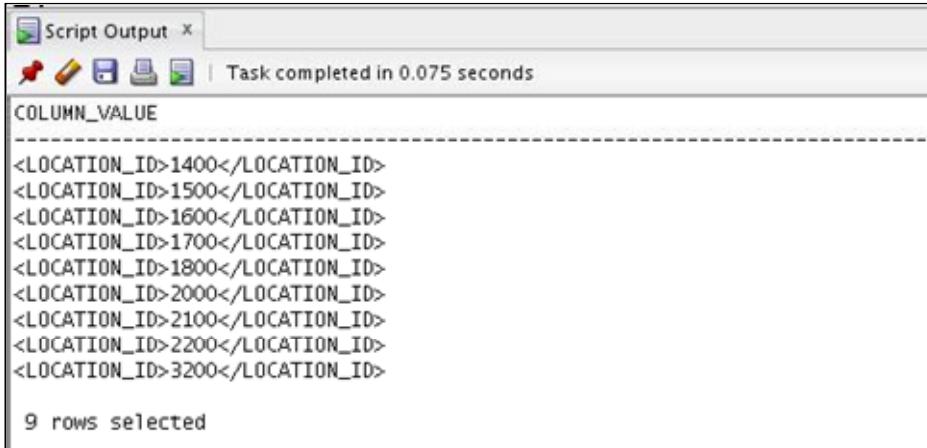
You can use the XMLTable SQL function to map the results of an XQuery evaluation into relational rows and columns of a new, virtual table. You can then insert the virtual table into a pre-existing database table, or you can query it by using SQL in a join expression. You can use XMLTable in the FROM clause of a SQL query expression.

Parameters for XMLTable

- XQuery_string_literal: The complete XQuery expression
- XMLNAMESPACES **clause (optional)**: Contains namespace declarations referenced by XQuery_string_literal and by the XPath expression in the PATH clause inside the COLUMN clause
- Passing **clause**: One or more SQL expressions, each returning an XMLType instance. The result of evaluating each expression is bound to the corresponding identifier for the evaluation of XQuery_string.
- COLUMNS **clause (optional)**: Defines the columns of the virtual table to be created by XMLTable in the absence of which XMLTable returns a row with a single XMLType pseudocolumn named COLUMN_VALUE. The COLUMNS clause has the following subclauses:
 - FOR ORDINALITY: Specifies the column of generated row numbers
 - Datatype: You must specify the data type of each resulting column except the FOR ORDINALITY column.

XMLTable: Example

```
SELECT *
FROM
XMLTABLE ('fn:collection("oradb:/OE/WAREHOUSES/")/ROW/LOCATION_ID');
```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. The title bar says 'Script Output X'. Below it, there are icons for redo, undo, save, and others. A message 'Task completed in 0.075 seconds' is displayed. The main area shows the output of the XMLTable query:

```
COLUMN_VALUE
-----
<LOCATION_ID>1400</LOCATION_ID>
<LOCATION_ID>1500</LOCATION_ID>
<LOCATION_ID>1600</LOCATION_ID>
<LOCATION_ID>1700</LOCATION_ID>
<LOCATION_ID>1800</LOCATION_ID>
<LOCATION_ID>2000</LOCATION_ID>
<LOCATION_ID>2100</LOCATION_ID>
<LOCATION_ID>2200</LOCATION_ID>
<LOCATION_ID>3200</LOCATION_ID>

9 rows selected
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLTable query and the output of the query are shown in the slide.

XMLTable Versus XMLQuery: Example

```
SELECT *
FROM XMLTABLE (
  for $i in fn:collection("oradb:/HR/DEPARTMENTS") /ROW
  return
    <Department id="{$i/DEPARTMENT_ID}">
      <Employees>
        . . .
```

1

All Rows Fetched: 27 in 0.008 seconds

COLUMN_VALUE
1 <Department id="10"><Employees><FIRST_NAME>Jennifer</FIRST_NAME><LAST_NAME>Wh...
2 <Department id="20"><Employees><FIRST_NAME>Michael</FIRST_NAME><LAST_NAME>Ha...
3 <Department id="30"><Employees><FIRST_NAME>Den</FIRST_NAME><LAST_NAME>Rapha...
26 <Department id="260"><Employees></Employees></Department>
27 <Department id="270"><Employees></Employees></Department>

```
SELECT XMLQUERY(
  for $i in fn:collection("oradb:/HR/DEPARTMENTS") /ROW
  return
  . . .
```

2

XMLQUERY("FOR\$INFN:COLLECTION("ORADB:/HR/DEPARTMENTS")/ROWRETURN<DEPARTMENTID="{\$I/DEPA...
1 <Department id="10"><Employees><FIRST_NAME>Jennifer</FIRST_NAME><LAST_NAME>Whalen</LAST_N...

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The examples in the slide show the differences between calling the SQL/XML functions XMLQuery and XMLTable. XMLQuery can be called directly in the SELECT clause, whereas XMLTable is called from the FROM clause of a SELECT statement. There is also a difference in the output that is generated. XMLQuery generates output in XML format, whereas XMLTable presents data as a virtual relational table.

The result of the first code example in the slide is a sequence of DEPARTMENT elements. XMLTable produces a virtual relational table whose rows are the DEPARTMENT elements.

Example 1 produces the same result as the XMLQuery example, except that each DEPARTMENT element is presented as a separate row. The value of the virtual column COLUMN_VALUE for each virtual table row is an XML fragment with a single DEPARTMENT element.

In this second code example in the slide, the XMLQUERY function is used to return the relational data as XML. In this case, all DEPARTMENT elements are presented as a single row.

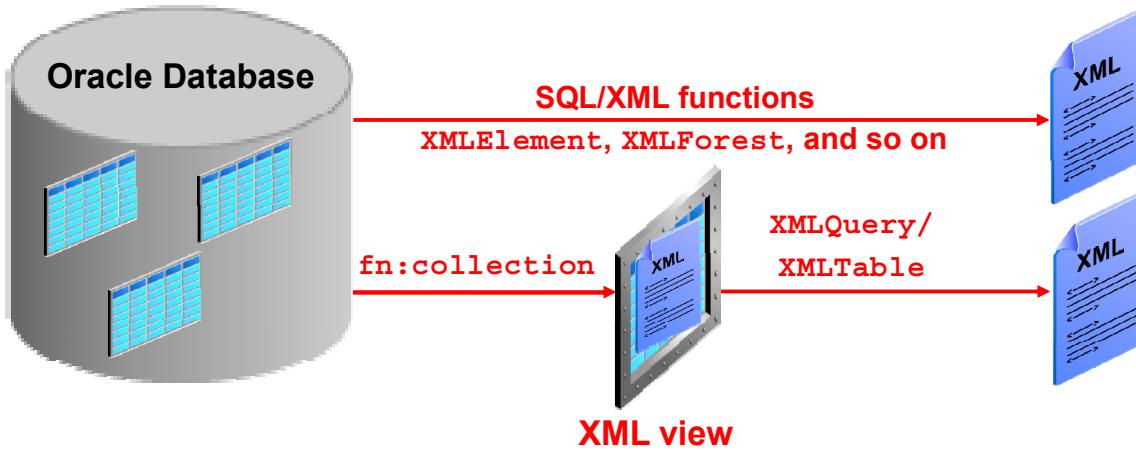
Lesson Agenda

- Retrieving XML content
- Using FLWOR expressions: Review
- Using XQuery functions in Oracle Database
- **Querying the database: Relational data**
- Querying the database: XMLType data
- Querying XMLType data by using SQL/XML standard functions
- Querying an XML document in the Oracle XML DB Repository
- Using a namespace with XQuery
- XQuery support in SQL*Plus: the XQUERY command



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Querying the Database: Relational Data



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can divide the XML structure into SQL structures and store the data in tables. Storing data in object-relational tables provides greater efficiency of space (there is no need to store tags) and time (SQL indexing techniques are mature and robust, so it may be faster to query data stored as SQL) than storing data using the binary XML storage model. Later, you can regenerate XML from these tables for data exchange.

Oracle Database provides SQL/XML generation functions, such as XMLElement, XMLForest, XMLConcat, XMLAgg, XMLATTRIBUTES, and so on. These functions convert user-defined types (UDTs) to their canonical XML formats. The SQL/XML generation functions are discussed in detail in the lesson titled "Transforming and Validating XML Data."

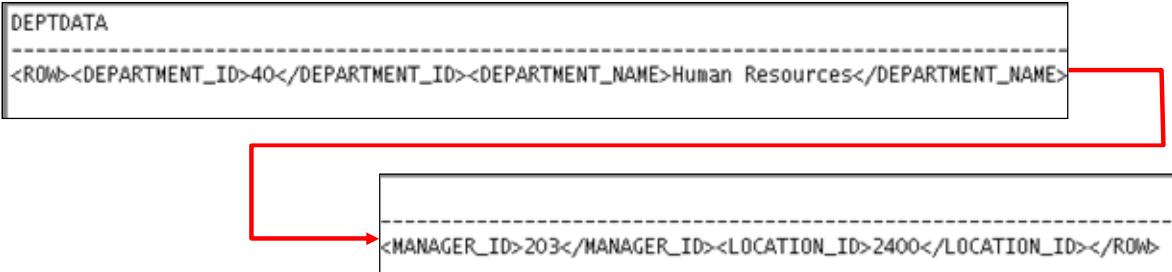
In Oracle Database, you can use XQuery functions `fn:doc` and `fn:collection` to query resources in Oracle XML DB Repository. In this lesson, the XQuery function `fn:collection` is used to query data in database tables and views.

Using XMLQuery to Query Relational Data

- Example:

```
SELECT XMLQuery (
  'for $i in fn:collection("oradb:/HR/DEPARTMENTS")
   where $i/ROW/DEPARTMENT_NAME= "Human Resources"
   return $i'
  RETURNING CONTENT) AS deptdata
FROM DUAL;
```

- Result (one line):



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use the XMLQuery SQL function to construct or query XML data. You can pass an XMLType column, table, or view as the context-item argument to XMLQuery. To query a relational table or view as if it were XML, without having to first create a SQL/XML view over it, you can use the XQuery function `fn:collection` within an XQuery expression.

The example in the slide uses the XMLQuery function `fn:collection` in a FLWOR expression to query the HR.DEPARTMENTS table.

Operations performed by the three FLWOR clauses in the example are as follows:

- `for` iterates over the sequences of XML elements that are returned by a call to `fn:collection`. Each element corresponds to a row of the DEPARTMENTS relational table and is bound to the `$i` variable. Because DEPARTMENTS is not an XMLType table, the top-level element corresponding to a row in this table is `ROW`.
- `where` filters the rows from the DEPARTMENTS table, keeping only those rows whose `DEPARTMENT_NAME` is *Human Resources*.
- `return` returns the filtered rows from the DEPARTMENTS table as an XML document containing XML fragments with `ROW` as their top-level element.

Joining Tables by Using XMLQuery

```
SELECT XMLQUERY(
'for $i in fn:collection("oradb:/HR/EMPLOYEES") /ROW
return
<EMPLOYEE id ="{$i/EMPLOYEE_ID/text() }">
{
    for $j in fn:collection("oradb:/HR/DEPARTMENTS") /ROW
    where $j/DEPARTMENT_ID eq $i/DEPARTMENT_ID
    return <DEPARTMENT>
    { $j/DEPARTMENT_NAME,
      $j/LOCATION_ID
    }</DEPARTMENT>
}</EMPLOYEE>
RETURNING CONTENT) as XML FROM DUAL;
```

XML

```
<EMPLOYEE id="198"><DEPARTMENT><DEPARTMENT_NAME>Shipping</DEPARTMENT_NAME><LOCATION_ID>1500</LOCATION_ID></DEPARTMENT></EMPLOYEE>
```

```
<EMPLOYEE id="199"><DEPARTMENT><DEPARTMENT_NAME>Shipping</DEPARTMENT_NAME><LOCATION_ID>1500</LOCATION_ID></DEPARTMENT></EMPLOYEE>
```

```
<EMPLOYEE id="200"><DEPARTMENT><DEPARTMENT_NAME>Administration</DEPARTMENT_NAME><LOCATION_ID>1700</LOCATION_ID></DEPARTMENT></EMPLOYEE>
```

...



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide uses nested FLWOR expressions. The query demonstrates a way to obtain an XML document containing data from multiple relational tables. The XQuery joins the EMPLOYEES table with the DEPARTMENTS table. Both tables are in the HR sample database schema.

Operations performed by the three FLWOR clauses in the example are as follows:

- The outer `for` iterates over the sequence of XML elements returned by `fn:collection`. Each element corresponds to a row of the EMPLOYEES relational table and is bound to the `$i` variable. Because EMPLOYEES is not an XMLType table, the top-level element corresponding to a row is `ROW`. The iteration over the `row` elements is unordered.
- The inner `for` iterates over a sequence of XML elements returned by `fn:collection`. Each element corresponds to a row of the DEPARTMENTS relational table and is bound to the `$j` variable.
- `where` filters the tuples (`$i, $j`), keeping only those whose `DEPARTMENT_ID` child is the same for `$i` and `$j`.

Lesson Agenda

- Retrieving XML content
- Using FLWOR expressions: Review
- Using XQuery functions in Oracle Database
- Querying the database: Relational data
- **Querying the database: XMLType data**
- Querying XMLType data by using SQL/XML standard functions
- Querying an XML document in the Oracle XML DB Repository
- Using a namespace with XQuery
- XQuery support in SQL*Plus: the XQUERY command



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Querying the Database: XMLType Data

```
-- create and populate the emp_resumes table.  
CREATE TABLE emp_resumes (  
    employee_id    NUMBER(6) PRIMARY KEY,  
    resume XMLType);  
/  
INSERT INTO emp_resumes VALUES (100,  
    XMLType(  
        '<?xml version="1.0"?>  
        <RESUME>  
            <FULL_NAME>Steven King</FULL_NAME>  
            <JOB_HISTORY>  
                <JOB_ID>AD_PRES</JOB_ID>  
            </JOB_HISTORY>  
        </RESUME>'));  
/  
  
-- Select the resume XMLType column from the emp_resumes table.  
SELECT resume  
FROM emp_resumes  
WHERE employee_id = 100;  
/
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can query XML data from XMLType columns in the following ways:

- Select XMLType columns in SQL, PL/SQL, or Java.
- Query XMLType columns by using XMLType methods.
- Use the Oracle Text operators to query the XML content.
- Use the XQuery language.

You can query XMLType data and extract portions of it using SQL/XML standard functions such as XMLQuery, XMLTable, XMLExists, and XMLCast. An XMLType column, such as resume, can be queried by using a standard SQL SELECT statement. The result of the example in the slide is:

```
table EMP_RESUMES created.  
1 rows inserted.  
RESUME  
-----  
<?xml version="1.0"?>  
<RESUME>  
    <FULL_NAME>Steven King</FULL_NAME>  
    <JOB_HISTORY>  
        <JOB_ID>AD_PRES</JOB_ID>  
    </JOB_HISTORY>  
</RESUME>
```

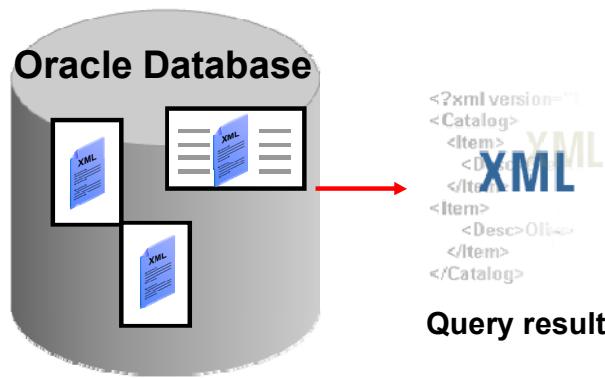
Querying the Database: XMLType Data

- Select XMLType columns:

```
SELECT resume
FROM emp_resumes
WHERE employee_id = 100;
```

- Use the SQL/XML functions:

- XMLEXISTS
- XMLCAST
- XMLTABLE
- XMLQUERY



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can query XML data from XMLType columns in the following ways:

- Select XMLType columns in SQL, PL/SQL, or Java.
- Query XMLType columns by using XMLType methods.
- Use the Oracle Text operators to query the XML content.
- Use the XQuery language.

You can query XMLType data and extract portions of it using SQL/XML standard functions such as XMLQuery, XMLTable, XMLExists, and XMLCast.

An XMLType column, such as `resume`, can be queried by using a standard SQL SELECT statement. The result of the example in the slide is:

```
RESUME
-----
<?xml version="1.0"?>
<RESUME>
  <FULL_NAME>Steven King</FULL_NAME>
  <JOB_HISTORY>
    <JOB_ID>AD_PRES</JOB_ID>
  </JOB_HISTORY>
</RESUME>
```

Note: Oracle Text operators facilitate the searching of XML documents stored in XMLType.

Note: The `existsNode`, `extractValue`, and `extract` functions are deprecated in Oracle Database 11g Release 2.

The preceding three Oracle functions use a subset of the W3C XPath recommendation to navigate the document. You can use each of the preceding functions either as a SQL function or as an XMLType member function. In the latter case, a correlation variable is required to execute the XMLType member function. For example, `e` is the correlation variable in the query:

```
SELECT XMLQuery(
  'for $i in fn:collection("oradb:/HR/EMP_RESUMES") //RESUME/FULL_NAME/text()
  return $i'
  RETURNING CONTENT) AS "Name"
FROM DUAL;
```

```
Name
-----
Steven King
```

It is recommended to use `XMLExists` in place of `existsNode` and `XMLCast` in place of `extractValue`.

You can also run the following code to obtain the same results as in the last slide:

```
SET LONG 5000
SELECT e.resume.getBlobVal() AS RESUME
FROM emp_resumes e
WHERE employee_id = 100;
```

```
RESUME
-----
<?xml version="1.0"?>
<RESUME>
  <FULL_NAME>Steven King</FULL_NAME>
  <JOB_HISTORY>
    <JOB_ID>AD_PRES</JOB_ID>
  </JOB_HISTORY>
</RESUME>

SQL> ■
```

Lesson Agenda

- Retrieving XML content
- Using FLWOR expressions: Review
- Using XQuery functions in Oracle Database
- Querying the database: Relational data
- Querying the database: XMLType data
- Querying XMLType data by using SQL/XML standard functions
- Querying an XML document in the Oracle XML DB Repository
- Using a namespace with XQuery
- XQuery support in SQL*Plus: the XQUERY command



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Querying an XMLType Table With XMLQuery and fn:collection

```
SELECT
XMLQuery('fn:collection("oradb:/HR/EMP_DEPARTMENTS")'
RETURNING CONTENT) AS EMP_DEPT_INFO
FROM DUAL;
```

```
EMP_DEPT_INFO
-----
<departments>
  <department num="1">
    <department_id>10</department_id>
    <department_name>Administration</department_name>
  </department>
  <department num="2">
    <department_id>20</department_id>
    <department_name>Marketing</department_name>
  </department>
  <department num="3">
    <department_id>30</department_id>
    <department_name>Purchasing</department_name>
  </department>
  <department num="4">
    <department_id>40</department_id>
    <department_name>Human Resources</department_name>
  </department>
</departments>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide uses the `fn:collection` function to query XML data from an XMLType table.

The `fn:collection` function is used in the example to create a view at run time containing data from the XMLType table `EMP_DEPARTMENTS`. The view is then queried by `XMLQuery`.

The output of the code example is shown in the slide using the Run Script [F5] in SQL Developer.

Note: The `EMP_DEPARTMENTS` table was created earlier in this lesson.

Querying an XMLType Table With XMLQuery and OBJECT_VALUE

```
SELECT XMLQuery(
    'for $i in /departments
     return $i'
    PASSING OBJECT_VALUE
    RETURNING CONTENT) AS EMP_DEPT_INFO
FROM EMP_DEPARTMENTS
```

```
Connected
EMP_DEPT_INFO
-----
<departments>
  <department num="1">
    <department_id>10</department_id>
    <department_name>Administration</department_name>
  </department>
  <department num="2">
    <department_id>20</department_id>
    <department_name>Marketing</department_name>
  </department>
  <department num="3">
    <department_id>30</department_id>
    <department_name>Purchasing</department_name>
  </department>
  <department num="4">
    <department_id>40</department_id>
    <department_name>Human Resources</department_name>
  </department>
</departments>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide uses the OBJECT_VALUE clause method to query XML data from an XMLType table.

The OBJECT_VALUE clause is used to hold the XMLType data that is queried by XQuery. In the example in the slide, the PASSING OBJECT_VALUE clause passes the data from the XMLType table EMP_DEPARTMENTS (referred in the FROM clause of the SELECT query) to the XQuery.

Note: To retrieve an entire XML document by using OBJECT_VALUE:

Use the OBJECT_VALUE pseudocolumn as an alias for the value of an object table. For an XMLType table that consists of a single column of XMLType, the entire XML document is retrieved.

If you use SQL*Plus, include the PAGESIZE and LONG settings to ensure that the entire XML document is printed correctly, without line breaks (as in the following example):

```
SET LONG 10000
SET PAGESIZE 100
SELECT OBJECT_VALUE
FROM purchaseorder;
```

Using XMLTable with PASSING and COLUMNS Clauses

```
SELECT xtab.poref, xtab.usr, xtab.requestor
FROM purchaseorder,
XMLTable(
  'for $i in /PurchaseOrder
   where $i/CostCenter eq "A10"
   return $i'
  PASSING OBJECT_VALUE
  COLUMNS
    poref VARCHAR2(20) PATH 'Reference',
    usr    VARCHAR2(20) PATH 'User'
                           DEFAULT 'Unknown',
    requestor VARCHAR2(20) PATH 'Requestor'
  ) xtab;
```

POREF	USR	REQUESTOR
JCHEN-20021009123337	JCHEN	John Z. Chen
JCHEN-20021009123337	JCHEN	John Z. Chen
...		
SKING-20021009123337	SKING	Steven A. King
SMCCAIN-200210091233	SMCCAIN	Samuel B. McCain
32 rows selected		



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows how to decompose the XML data in the XMLType table PURCHASEORDER into an XMLType collection element xtab by using the XMLTable clauses PASSING and COLUMNS. The query returns the values of the <Reference>, <User>, and <Requestor> nodes for all purchase orders with CostCenter A10.

In the example, you use the OBJECT_VALUE pseudocolumn in the PASSING clause of XMLTable to pass the PURCHASEORDER table as a context item to the XQuery expression.

The COLUMNS clause defines the columns of the virtual table to be created by XMLTable. In the example, XMLTable decomposes the result of the XQuery evaluation into three VARCHAR2 columns (poref, usr, and requestor) of the xtab virtual table.

Querying an XMLType Table by Using XMLTable

You can query XML data by joining a virtual table and a database table.

```
SELECT xtab.column_value
  FROM purchaseorder p,
       XMLTable('for $i in /PurchaseOrder
                 where $i/CostCenter eq "A10"
                 return
                   <A10po pono="{\$i/Reference}" />'
                PASSING p.OBJECT_VALUE
      ) xtab;
```

COLUMN_VALUE
<A10po pono="JCHEN-20021009123337123PDT"></A10po>
<A10po pono="JCHEN-20021009123337223PDT"></A10po>
<A10po pono="SKING-20021009123337153PDT"></A10po>

<A10po pono="SKING-20021009123337503PDT"></A10po>
<A10po pono="SMCCAIN-20021009123337403PDT"></A10po>

32 rows selected



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example shows how to use the `XMLTable` construct to query an `XMLType` table. The query performs a join between the `xtab` virtual table and the `PURCHASEORDER` database table.

It uses `XMLTable` to query the `XMLType` table `PURCHASEORDER`. To provide the context item (`PURCHASEORDER` table, in the example) for `XMLTable`, the `PASSING` clause is used. The `COLUMN_VALUE` pseudocolumn of the resulting virtual table holds a constructed element `A10po`. This constructed element contains Reference information for those purchase orders whose `CostCenter` element has the value `A10`.

Querying an XMLType Column by Using XMLQuery

```

SELECT warehouse_name,
       XMLQuery(
         'for $i in /Warehouse
          where  $i/Area > 10000
          return <Details>
            <Docks num="{$i/Docks}" />
            <Rail>
              {if ($i/RailAccess = "Y")
                then "true" else "false"
              }
            </Rail>
          </Details>' PASSING warehouse_spec
  RETURNING CONTENT) big_warehouses
FROM warehouses;

```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can query an XMLType column by using the XQuery functions XMLQuery and XMLTable.

In the slide example, the IF statement is used to check for warehouses with railroad access. The XMLType column (WAREHOUSE_SPEC) is passed in through the PASSING clause as a context item.

WAREHOUSE_NAME	BIG_WAREHOUSES
Southlake, Texas	<Details><Docks num="2"></Docks><Rail>false</Rail></Details>
San Francisco	<Details><Docks num="1"></Docks><Rail>false</Rail></Details>
New Jersey	<Details><Docks num=""></Docks><Rail>false</Rail></Details>
Seattle, Washington	<Details><Docks num="3"></Docks><Rail>true</Rail></Details>
Toronto	
Sydney	
Mexico City	
Beijing	
Bombay	
9 rows selected	

Querying XMLType Data by Using XMLEXISTS

Use the XML EXISTS function in a WHERE SQL clause:

-- Run this script using SQL*Plus in a terminal window.

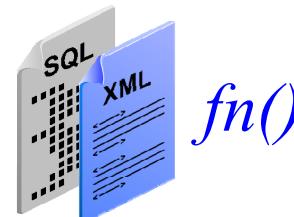
```
SELECT OBJECT_VALUE
FROM purchaseorder
WHERE
    XMLExists('/PurchaseOrder[SpecialInstructions="Expedite"]'
PASSED OBJECT VALUE);
```

```
OBJECT_VALUE
-----
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames

OBJECT_VALUE
-----
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames

13 rows selected.

SQL>
```



SQL/XML standard function

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

`XMLExists` checks whether a given XQuery expression returns a nonempty XQuery sequence. `XMLExists` returns a Boolean value `TRUE` if the result of evaluating the XQuery expression is a nonempty XQuery sequence. Otherwise, the function returns `FALSE`.

Syntax: XMLEXISTS (XQuery string [XML passing clause])

The slide example uses the XMLExists SQL/XML standard function to select rows with SpecialInstructions set to Expedite. The query returns complete purchase order documents.

The `XMLExists` SQL/XML standard function is similar to the `existsNode` Oracle function, but it differs in the following ways:

- `XMLExists` accepts an arbitrary XQuery expression (possibly including a prolog); `existsNode` accepts only an XPath expression.
 - `XMLExists` returns a Boolean value, TRUE or FALSE; `existsNode` returns 1 or 0.
 - `XMLExists` tests whether its XQuery-expression argument returns a nonempty sequence. `existsNode` tests whether its XPath-expression argument targets at least one element node or text node. The set of XPath expressions is a proper subset of the XQuery expressions.

It is recommended that you use `XMLExists` instead of `existsNode`.

XPath expressions can contain predicates. Therefore, `XMLExists` can determine whether or not a given node exists in the document, and whether or not a node with a specified value exists in the document.

You can create function-based indexes by using the `XMLExists` SQL/XML function to speed up the execution. You can also create an `XMLIndex` index to help speed up arbitrary XQuery searching.

Note: You can use the `XMLExists` function in a SQL `WHERE` clause or `CASE` expression. If you want to use it in a `SELECT` list, wrap it in a `CASE` expression, as in the following example:

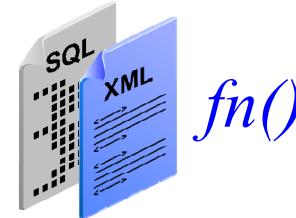
```
CASE WHEN XMLExists(...) THEN 'TRUE' ELSE 'FALSE' END
```

Using the XMLCAST Function

Use XMLCast to cast items from XML to SQL scalar data type.

```
SELECT XMLCast (XMLQuery('/PurchaseOrder/Reference' PASSING
    OBJECT_VALUE
    RETURNING CONTENT)
AS VARCHAR2(100)) "REFERENCE"
FROM purchaseorder
WHERE
    XMLExists('/PurchaseOrder[SpecialInstructions="Expedite"]' 
PASSING OBJECT_VALUE);
```

```
Script Output x
Task completed in 0.022 seconds
Connected
REFERENCE
JCHEN-20021009123337123PDT
AMCEWEN-20021009123336271PDT
SKING-20021009123336321PDT
AWALSH-20021009123337303PDT
SKING-20021009123338294PDT
WSMITH-20021009123338154PDT
TFOX-20021009123337463PDT
AWALSH-20021009123336642PDT
SKING-20021009123336622PDT
SKING-20021009123336822PDT
AWALSH-20021009123336101PDT
WSMITH-20021009123336412PDT
AWALSH-20021009123337954PDT
13 rows selected
```



SQL/XML standard function

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLCast function, introduced in Oracle Database 11g, allows you to cast items from XML to SQL scalar data type. The first argument is a SQL expression that is evaluated. You can specify NUMBER, VARCHAR2, and any of the date and time data types as the second argument.

Syntax

```
XMLCAST ( value_expression AS datatype )
```

The result of evaluating the first XMLCast argument is an XML value. It is converted to the target SQL data type by using the XQuery atomization process, and then casting the XQuery atomic values to the target data type. If this conversion fails, an error is raised. If conversion succeeds, the result returned is an instance of the target data type.

Note: Unlike the SQL/XML standard, Oracle XML DB limits the use of XMLCast to cast XML to a SQL scalar data type. Oracle XML DB does not support casting XML to XML or from a scalar SQL type to XML.

The slide example shows how to extract the scalar value of an XML fragment by using XMLCAST. The query extracts the scalar value of the Reference node.

The XMLCast SQL/XML standard function is similar to the extractValue Oracle function, but it differs in that extractValue does not allow or require you to specify a target data type. With XMLCast you have the advantage of control over the data type, in addition to portability. If the SQL scalar data type cannot be determined at compile time, extractValue returns a value of type VARCHAR2(4000), which might not always be what you expect or want. You can work around this obstacle by using the cast SQL function, but XMLCast is a better choice in this case.

Using XMLCAST and XMLQuery: Example

```
SELECT XMLCast(XMLQuery('$/PurchaseOrder/Reference'
    PASSING OBJECT_VALUE AS "p"
    RETURNING CONTENT)
    AS VARCHAR2(30))
FROM purchaseorder p, hr.employees e
WHERE XMLCast(XMLQuery('$/PurchaseOrder/User'
    PASSING OBJECT_VALUE AS "p"
    RETURNING CONTENT)
    AS VARCHAR2(30)) = e.email
AND e.employee_id = 100;
```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the query executed and its results. The output window has a toolbar at the top with icons for script, edit, file, and help. Below the toolbar, it says 'Task completed in 0.029 seconds'. The message 'Connected' is followed by the query: 'XMLCAST(XMLQUERY('\$/PURCHASEORDER/REFERENCE'PASSINGOBJECT_VALUEAS"P"RETURNINGCONTENT)ASVARCHAR2(30))'. The results are listed below, showing 13 rows selected, each consisting of a timestamp string starting with 'SKING-' followed by a sequence of digits and letters.

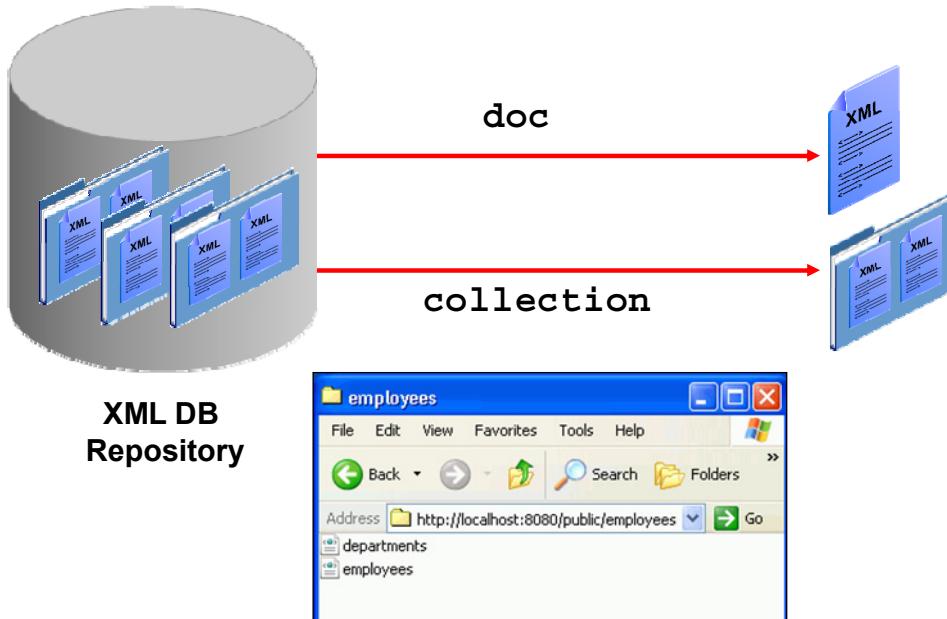
Result
SKING-20021009123337153PDT
SKING-20021009123336952PDT
SKING-20021009123335560PDT
SKING-20021009123336321PDT
SKING-20021009123337974PDT
SKING-20021009123338294PDT
SKING-20021009123337703PDT
SKING-20021009123337383PDT
SKING-20021009123337503PDT
SKING-20021009123336622PDT
SKING-20021009123336822PDT
SKING-20021009123336131PDT
SKING-20021009123336392PDT



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example shows how to perform a join based on the values of a node in an XML document and data in another table.

Querying an XML Document in the Oracle XML DB Repository



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle XML DB supports the following XQuery functions for querying files from the repository:

- doc: Queries files from the repository
- collection: Maps to directories

The `doc` function is an XQuery standard function that takes in a Uniform Resource Identifier (URI) and returns a document node corresponding to that URI. In the server, this function uses the Oracle XML DB Repository. Thus, valid URIs to this function are those that map to a valid XML file in the Oracle XML DB Repository.

The `collection` function is similar to `doc`. But instead of working on a single XML document, `collection` works on repository folder resources. This function, therefore, takes in as parameter a valid URI that maps to a folder in the repository and returns a sequence of XML files that are present in that folder.

The graphic in the slide shows the two XML documents `employees.xml` and `departments.xml` in the `/public/employees` repository folder.

Using doc to Query Repository Documents

```
SELECT XMLQuery(
  'for $i in
    fn:doc("/public/employees/employees.xml")//EMPLOYEE
  order by $i
  return $i'
  RETURNING CONTENT) AS xml
FROM DUAL
```

The screenshot shows the Oracle SQL Developer interface with a 'Script Output' window. The window title is 'Script Output X'. It displays the results of an XQuery execution. The output starts with 'Task completed in 0.013 seconds' and then shows the XML structure of the employees from the employees.xml file. The XML is as follows:

```
<EMPLOYEE>
  <EMPNO>100</EMPNO>
  <ENAME>King</ENAME>
  <SAL>24000</SAL>
</EMPLOYEE>
<EMPLOYEE>
  <EMPNO>101</EMPNO>
  <ENAME>Kochhar</ENAME>
  <SAL>17000</SAL>
</EMPLOYEE>
<EMPLOYEE>
  <EMPNO>102</EMPNO>
  <ENAME>De Haan</ENAME>
  <SAL>17000</SAL>
</EMPLOYEE>
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The `doc` function can be used in XQuery to query an XML file stored in the Oracle XML DB Repository. The slide example contains an XQuery expression that extracts XML data from `employees.xml` by using the `doc` function, and then binds the XQuery variables to parts of that data by using the FLWOR expression's `for` clause. The `fn:doc` function takes a string that represents a repository file resource (URI) as parameter and returns the document targeted by that URI, which must be a file of well formed XML data.

The slide example demonstrates how you can use the XPath `//` construction in an XQuery expression.

Using collection to Query Repository Documents In a Folder

```
SELECT XMLQuery(
  'for $i in fn:collection("/public/employees")
   return $i//DEPARTMENT/DNAME'
  RETURNING CONTENT)AS XML
FROM DUAL
```



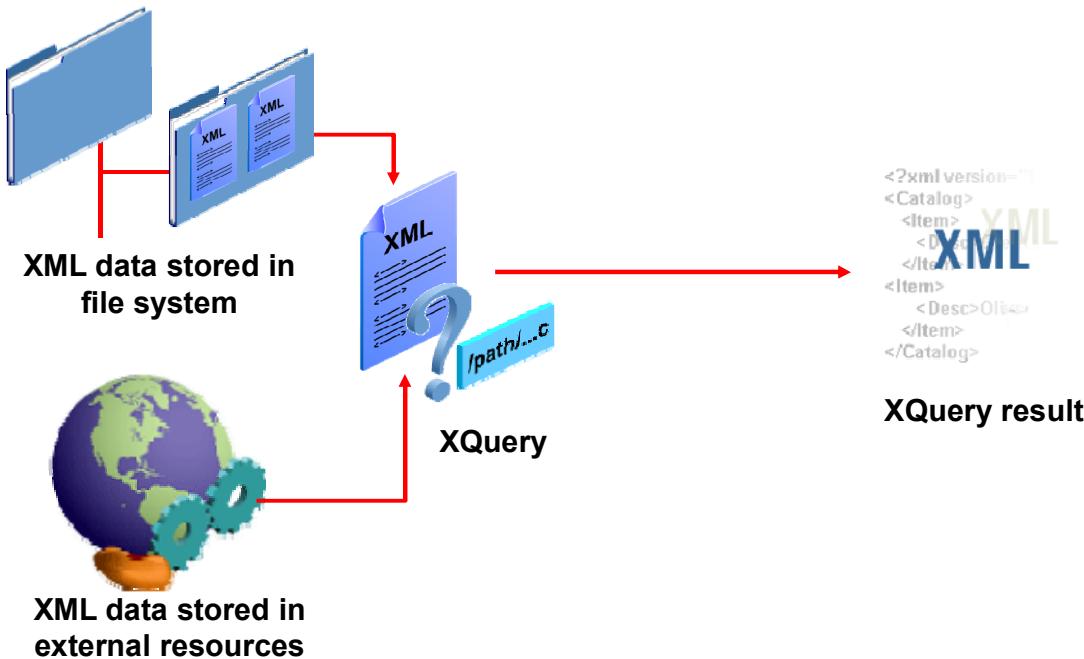
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Similar to doc, the collection function can also be used in XQuery to query an XML file that is stored in the Oracle XML DB Repository. However, unlike doc, collection works on the repository folder resources (each file in the folder must contain well formed XML data).

The example in the slide contains an XQuery expression that extracts XML data from the /public/employees repository folder. The collection function takes a string parameter that represents a resource (URI) for the /public/employees folder. Because this folder contains two XML files (employees.xml and departments.xml), the collection function returns both these documents, and then binds the XQuery variables to parts of that data by using the for clause of the FLWOR expression.

Using XQuery to Query Sources Beyond the Database



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

With XQuery, you can process XML documents from data stored in various places, such as a web site or file system, and even XML data that is transient.

Oracle XML DB provides very high performance and scalability for XML operations on data that is stored in the database. It is advisable to move XML data into the database. However, you may come across situations where you need to address both internal and external content.

You can also use XQuery to query an XML Really Simple Syndication (RSS) feed. Because an RSS feed is a hosted XML file from which your RSS newsreader pulls headlines or other content, you can handle it as you would handle any other XML document that is available through the web.

Using a Namespace with XQuery: Examples

```
SELECT * FROM XMLTable(
  XMLNAMESPACES('http://emp.com' AS "e"),
  'for $i in doc("/public/empsns.xml")
  return $i/e:emps/e:emp'
  COLUMNS name VARCHAR2(6) PATH '@ename',
  id NUMBER PATH '@empno');
```

1

```
SELECT * FROM XMLTable(
  'declare default element namespace "http://emp.com"; (:)
   for $i in doc("/public/empsns.xml")
   return $i/emps/emp'
  COLUMNS name VARCHAR2(6) PATH '@ename',
  id NUMBER PATH '@empno' )
```

2



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

An XML namespace, identified by an Internationalized Resource Identifier (IRI), is a case-sensitive string of characters identifying a resource. It provides a universally unique name for a collection of XML names that are made up of element and attribute names.

In an XQuery, you can refer to multiple XML documents. If you refer to an element that is present in multiple XML documents, you can use a namespace to resolve the naming conflict.

You can declare a namespace prefix in an XQuery by using a prolog namespace declaration before an XQuery expression. You can establish the namespace as the default namespace for the expression. An XQuery namespace declaration has no effect outside of its XQuery expression. To declare a namespace prefix for use in an XMLTable expression outside of the XQuery expression, use the XMLNAMESPACES clause. This clause also covers the XQuery expression argument to XMLTable, thereby eliminating the need for a separate declaration in the XQuery prolog.

Note: The two examples in the slide assume that the empsns.xml XML document is foldered into the Oracle XML DB Repository as /public/empsns.xml. This is shown in the code_08_48_as.sql script.

The contents of emps.xml are as follows:

```
<?xml version="1.0"?>
<emps xmlns="http://emp.com">
    <emp empno="1" deptno="10" ename="John" salary="21000"/>
    <emp empno="2" deptno="10" ename="Jack" salary="31000"/>
    <emp empno="3" deptno="20" ename="Jill" salary="100001"/>
</emps>
```

The examples in the slide use a namespace with XQuery to produce the following result:

Script Output	
	Task completed in 0.012 seconds
<hr/>	
NAME	ID
John	1
Jack	2
Jill	3

Example 1

This example uses XMLNAMESPACES to define the prefix “e” for the namespace `http://emp.com`. This namespace is used in the COLUMNS clause as well as the XQuery expression of the XMLTable function.

Example 2

You can also define a default namespace by using the prolog declaration. The empty comments (`:`) at the end of each XQuery statement are introduced to stop the SQL*Plus interpreter from treating the `;` character as the end of the SQL statement.

XQuery Support in SQL*Plus: XQUERY Command

```
[oracle@EDRSR13P1 ~]$sqlplus hr/hr
SQL*Plus: Release 11.2.0.1.0 Production on Tue Feb 1 10:29:21 2011
Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> set long 20000
SQL> xquery for $i in fn:collection("oradb:/HR/DEPARTMENTS") return $i
2 /
Result Sequence
-----
<ROW><DEPARTMENT_ID>10</DEPARTMENT_ID><DEPARTMENT_NAME>Administration</DEPARTMENT_NAME><MANAGER_ID>200</MANAGER_ID><LOCATION_ID>1700</LOCATION_ID></ROW>

<ROW><DEPARTMENT_ID>20</DEPARTMENT_ID><DEPARTMENT_NAME>Marketing</DEPARTMENT_NAME><MANAGER_ID>201</MANAGER_ID><LOCATION_ID>1800</LOCATION_ID></ROW>
```



```
Result Sequence
-----
<ROW><DEPARTMENT_ID>270</DEPARTMENT_ID><DEPARTMENT_NAME>Payroll</DEPARTMENT_NAME><LOCATION_ID>1700</LOCATION_ID></ROW>

27 item(s) selected.
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Starting with Oracle Database 11g, you can create and test XQuery statements in SQL*Plus.

You can enter an XQuery statement at the SQL prompt by preceding the expression with the SQL*Plus XQUERY command. Termination of the XQuery statement is signified by a slash (/).

Oracle Database treats the XQuery expressions that are submitted with this command in the same way it treats XQuery expressions in the XMLQuery and XMLTable SQL functions. Both command-line XQuery and SQL functions provide identical execution optimization.

Note: There are a few SQL*Plus SET commands that you can use for settings that are specific to XQuery. Use SHOW XQUERY to see the current settings. For more information about XQuery support in SQL*Plus, see the *Oracle XML DB Developer's Guide, 12c Release 1 (12.1)*.

Quiz

You can use the XQuery functions `fn:doc` and `fn:collection` to either query resources in Oracle XML DB Repository or query data in database tables and views.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a

Quiz

XMLQuery can be called directly in the SELECT clause, whereas XMLTable is called from the FROM clause of a SELECT statement. There is also a difference in the output that is generated. XMLQuery generates output in XML format, whereas XMLTable presents data as a virtual relational table.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a

Summary

In this lesson, you should have learned how to:

- Explain the different types of XML queries
- Describe XQuery support in Oracle XML DB
- Query a relational table as if it were XML data
- Use SQL/XML standard functions and XQuery to retrieve XML data that is stored in `XMLType` tables
- Query an XML document in the Oracle XML DB Repository
- Use a namespace with XQuery
- Use the XQuery `XQUERY` command in SQL*Plus



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In this lesson, you learned:

- To retrieve XML data
- How XQuery is supported in Oracle XML DB
- To query a relational table as if it were XML data
- To query native `XMLType` data

You learned how to retrieve XML data by using the following SQL/XML standard functions:

- `XMLQuery`
- `XMLTable`
- `XMLExists`
- `XMLCast`

You learned how to use the `fn:collection` Xquery function in association with the `XMLTable` and `XMLQuery` functions to extract data that is stored in relational tables and to extract native `XMLType` data.

Practice 8-1: Overview

This practice covers the following topics:

- Querying XMLType columns
- Retrieving XML data by using the following SQL/XML standard functions:
 - XMLQuery
 - XMLTable
 - XMLExists
 - XMLCast



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 8-2: Overview

This practice covers the following topics:

- Using XQuery to query XML data from various resources
- Using the SQL*Plus XQUERY command to query XML data



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Updating XML Content Using XQuery Update

9

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Update an entire XML document
- Replace XML nodes
- Use SQL UPDATE and XQuery Update
- Update XML data to NULL values
- Insert an element into a collection
- Delete XML nodes
- Create a view by using updated XML data
- Use XQuery Update to transform the XML data



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XQuery Update

Oracle XML DB supports the following W3C XQuery standards:

- XQuery 1.0 Recommendation
- XQuery Update Facility 1.0 Recommendation
- XQuery and XPath Full Text 1.0 Recommendation



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XQuery is the W3C language designed for querying and updating XML data. Oracle XML DB support for the XQuery language has been updated to reflect the latest version of the XQuery standard, W3C XQuery 1.0 Recommendation. Oracle XML DB supports the following W3C XQuery standards:

- XQuery 1.0 Recommendation
- XQuery Update Facility 1.0 Recommendation
- XQuery and XPath Full Text 1.0 Recommendation

The XQuery Update feature is available in Oracle Database starting with version 11.2.0.3.0. For detailed information on XQuery and XPath, see the *Oracle XML DB Developer's Guide 12c Release 1 (12.1)* documentation guide or attend the Oracle University XML Fundamentals course.

Migrating from Oracle Functions for Updating XML Data to XQuery Update

- The XQuery Update Facility 1.0 Recommendation is supported by Oracle XML DB starting with Oracle Database 12c Release 1 (12.1).
- XQuery Update is an extension to the W3C XQuery standard.
- You can use XQuery Update operations to update XML content replacing either the entire content of a document or parts of a document.
- In Oracle XML DB, you execute XQuery operations by using the `XMLQuery` operator.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XQuery Update Facility 1.0 Recommendation is supported by Oracle XML DB starting with Oracle Database 12c Release 1 (12.1).

XQuery Update is an extension to W3C XQuery standard that makes it possible to update the content of XML documents. XQuery Update operations can modify the values of existing nodes, replace a fragment of XML with another fragment of XML and insert and remove nodes from the document. In Oracle XML DB, you execute XQuery operations by using the `XMLQuery` Operator.

Note

- For a list of the deprecated Oracle-specific SQL functions that were used to update XML data prior to Oracle Database 12c (12.1), see the *Oracle XML DB Developer's Guide 12c Release 1 (12.1)* documentation guide.
- If you have legacy code that uses these functions, Oracle recommends that you migrate that code to use XQuery Update. The *Oracle XML DB Developer's Guide 12c Release 1 (12.1)* documentation guide provides information about which XQuery Update constructs you can use to replace the use of the Oracle-specific XML updating functions in queries.

XQuery Update Snapshots

- An XQuery expression can require more than one update operation.
- The unit of change is therefore an entire XQuery query.
- In the last step of XQuery expression evaluation, the update operations are applied in an atomic fashion, and the snapshot is terminated.
- The set of update operations used in a given query is not necessarily applied in the order written.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

An XQuery expression can require more than one update operation. For example, XQuery Update performs all such operations for the same query as an atomic operation: Either they all succeed or none of them do, if an error is raised.

The unit of change is therefore an entire XQuery query. To effect this atomic update behavior, before evaluating your query, XQuery Update takes a snapshot of the data (XDM instances) whose modification is called for by the query. It also adds the update operations called for by the query to the pending update list. The snapshot is an evaluation context for an XDM instance that is the update target.

As the last step of XQuery expression evaluation, the pending update list is processed, applying the indicated update operations in an atomic fashion, and terminating the snapshot.

Note that the atomic nature of snapshot semantics means that a set of update operations used in a given query is not necessarily applied in the order written. In fact, the order of applying update operations is fixed and specified by the XQuery Update feature standard. This means that an update operation does not see the result of any other update operation for the same query. There is no notion of an intermediate or interim update state; all updates for a query are applied together, atomically.

XQuery Update Snapshots: Example

```
SELECT XMLQuery ('copy $npo := $po modify
  (delete nodes
   $npo/PurchaseOrder/ShippingInstructions/address,
  delete nodes $npo/PurchaseOrder/ShippingInstructions [name =
  "Sarah J. Bell"]/address)
  return $npo' PASSING VALUE(po) AS "po" RETURNING CONTENT)
FROM purchaseorder po;
```

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-instance.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-annotation.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-attribute.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-complex-type.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-element.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-namespace.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-type.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-unique.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-union.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-annotation-processor.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-attribute-processor.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-complex-type-processor.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-element-processor.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-namespace-processor.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-type-processor.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-unique-processor.xsd"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/2001/XMLSchema-union-processor.xsd"?>
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://localhost:8080/source/schemas/poSource/xsd/purchaseorder.xsd">
  <Reference>SBELL-2002100912333601PDT</Reference>
  <Actions>
    <Action>
      <User>SVOLLMAN</User>
    </Action>
  </Actions>
  <Reject/>
  <Requestor>Sarah J. Bell</Requestor>
  <User>SBELL</User>
  <CostCenter>S30</CostCenter>
  <ShippingInstructions>
    <name>Sarah J. Bell</name>
    <telephone>650 506 7400</telephone>
  </ShippingInstructions>
</PurchaseOrder>
```

The address element is removed from the purchaseorder table



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The code example in the slide contains a query which demonstrates multiple updates. One of the updates is reading values which are modified by the other update. Due to the snapshot updates semantics, the second update does not see the change performed by the first update.

There are two delete expressions in the code example. The two delete expressions have to be based on snapshot semantics as defined in the XQuery Update Facility. The second delete expression is looking at an address element node which is deleted by the first delete expression. However, since the snapshot is at the top expression level, therefore, the first deletion is not applied immediately. Instead, it is deferred until the transform expression returns and thus the predicate in the second deletion is still evaluated.

Note: Issue a rollback after confirming the results to return the table to its default status.

Updating XML Data

The remainder of the lesson covers updating XML data, both transient data and data stored in tables. This includes:

- Updating an entire XML document
- Replacing XML nodes
- Using SQL UPDATE and XQuery Update
- Updating XML data to NULL values
- Inserting an element into a collection
- Deleting XML nodes
- Creating a view by using updated XML data



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The remainder of the lesson covers updating XML data, both transient data and data stored in tables. This includes:

- Updating an entire XML document using the SQL UPDATE statement
- Replacing XML nodes using the replace value of node with clause of the SQL/XML XMLQuery function
- Using SQL UPDATE and XQuery Update
- Updating XML data to NULL values using the SQL/XML XMLQuery function
- Using XQuery Update to create new views of XML data using the CREATE OR REPLACE VIEW statement with the SQL/XML XMLQuery function
- Inserting an element into a collection using the insert node into clause of the SQL/XML XMLQuery function
- Deleting XML nodes using the delete nodes clause of the SQL/XML XMLQuery function

Updating an Entire XML Document

- To update an entire XML document, you use a SQL UPDATE statement.
- The right side of the UPDATE statement SET clause must be an XMLType instance.
- You can create it in any of the following ways:
 - Use SQL functions or XML constructors that return an XML instance.
 - Use the PL/SQL DOM APIs for XMLType that change and bind an existing XML instance.
 - Use the Java DOM API that changes and binds an existing XML instance.
- You can make updates for non-schema-based documents stored as binary XML in a piecewise manner.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

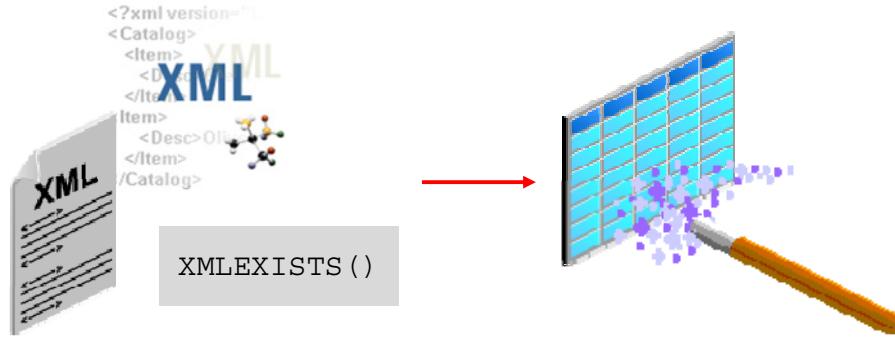
To update an entire XML document, use a SQL UPDATE statement. The right side of the UPDATE statement SET clause must be an XMLType instance. You can create it in any of the following ways:

- Use SQL functions or XML constructors that return an XML instance.
- Use the PL/SQL DOM APIs for XMLType that change and bind an existing XML instance.
- Use the Java DOM API that changes and binds an existing XML instance.

You can make updates for non-schema-based documents stored as binary XML in a piecewise manner.

General Syntax for an XQuery Update

```
UPDATE tablename  
SET XML = XMLQuery(XQuery-Update operation)  
WHERE XMLEXISTS()
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

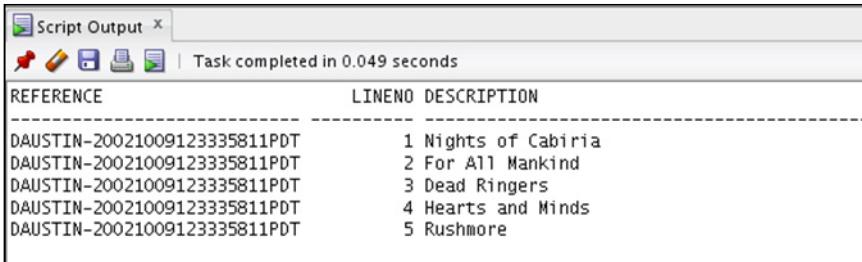
The general form of an XQuery Update is:

```
UPDATE tablename  
SET XML = XMLQuery(XQuery-Update operation)  
WHERE XMLEXISTS()
```

The XMLEXISTS operator is used to determine which documents the XQUERY Update operation is applied to. Predicates supplied to the XQuery Update operation determine which nodes within the documents selected by the XMLEXISTS operation are actually updated.

Example: Updating XMLType Data by Using SQL UPDATE (Current State)

```
SELECT t.reference, li.lineno, li.description
FROM purchaseorder po,
     XMLTable('$p/PurchaseOrder' PASSING po.OBJECT_VALUE AS "p"
              COLUMNS reference VARCHAR2(28) PATH 'Reference',
              Lineitem XMLType PATH 'LineItems/LineItem') t,
     XMLTable('$1/LineItem' PASSING t.lineitem AS "l"
              COLUMNS lineno NUMBER(10) PATH '@ItemNumber',
              description VARCHAR2(128) PATH 'Description') li
WHERE t.reference = 'DAUSTIN-20021009123335811PDT' AND
      ROWNUM < 6;
```



REFERENCE	LINENO	DESCRIPTION
DAUSTIN-20021009123335811PDT	1	Nights of Cabiria
DAUSTIN-20021009123335811PDT	2	For All Mankind
DAUSTIN-20021009123335811PDT	3	Dead Ringers
DAUSTIN-20021009123335811PDT	4	Hearts and Minds
DAUSTIN-20021009123335811PDT	5	Rushmore



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This slide example displays the current state of the XMLType table `purchaseorder`. The query uses virtual tables and to display the `Reference` and `Lineitem` elements and the `ItemNumber` attribute as columns in a relational table for Reference **DAUSTIN-20021009123335811PDT**. The result shows three rows. In the next example, we will update the XMLType data for the same XML document by using the SQL `UPDATE` statement.

Example: Updating XMLType Data Using SQL UPDATE (Updated State)

```
UPDATE purchaseorder po
SET po.OBJECT_VALUE = XMLType(bfilename('XML_DIR', 'NEW-
DAUSTIN-20021009123335811PDT.xml'),
nls_charset_id('AL32UTF8'))
WHERE XMLExists('$p/PurchaseOrder[Reference="DAUSTIN-
20021009123335811PDT"]'
PASSED po.OBJECT_VALUE AS "p");
```

1 rows updated.

REFERENCE	LINENO	DESCRIPTION
DAUSTIN-20021009123335811PDT	1	Dead Ringers
DAUSTIN-20021009123335811PDT	2	Getrud
DAUSTIN-20021009123335811PDT	3	Branded to Kill

The output reflect the contents of the XML document NEW-DAUSTIN-20021009123335811PDT.xml. The file contains (3) rows.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example updates an XMLType instance (the XML document that we just queried) by using a SQL UPDATE statement. If we reissue the SELECT statement from the previous example to check the current state of the updated and highlighted XML document, the second output in the slide now shows the result of the update operation.

Replacing XML Node Values (Current State)

```
SELECT XMLQuery('$/PurchaseOrder/Actions/Action[1] '
    PASSING po.OBJECT_VALUE AS "p"
    RETURNING CONTENT) Action
FROM purchaseorder po
WHERE XMLExists('$/PurchaseOrder [Reference="SBELL-
    2002100912333601PDT"] '
    PASSING po.OBJECT_VALUE AS "p") ;
```

```
ACTION
-----
<Action>
  <User>SVOLLMAN</User>
</Action>
```

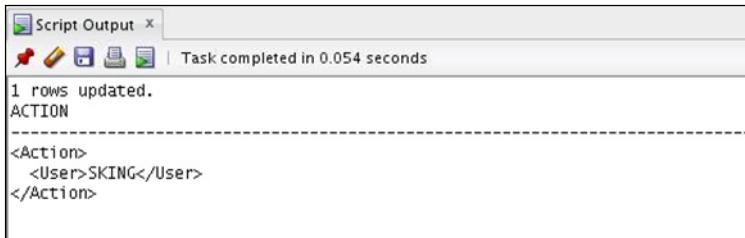


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the next slide example, some XML node values are replaced in an XML document. Before the replace operation is performed, this SELECT query is used to display the current state of the highlighted XML document. The result in the slide shows the user as SVOLLMAN.

Replacing XML Node Values (Updated State)

```
UPDATE purchaseorder po
SET po.OBJECT_VALUE =
  XMLQuery('copy $i := $p1 modify
            (for $j in $i/PurchaseOrder/Actions/Action[1]/User
             return replace value of node $j with $p2)
            return $i' PASSING po.OBJECT_VALUE AS "p1",
                     'SKING' AS "p2" RETURNING CONTENT)
WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-
2002100912333601PDT"]' PASSING po.OBJECT_VALUE AS "p");
```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the following output:

```
Script Output x
Task completed in 0.054 seconds
1 rows updated.
ACTION
-----
<Action>
  <User>SKING</User>
</Action>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example uses XQuery Update on the right side of a SQL UPDATE statement to update an existing (highlighted) XML document instead of creating a new document. The entire document, not just the selected part, is updated.

If you reissue the SELECT statement from the previous example, the output shows the results of the update operation. The user is now SKING instead of SVOLLMAN.

In this example, we pass the literal string SKING to the XQuery expression as a variable (\$p2). In this simple example, because the value is a literal string, you could have simply used the replace value of node \$j with SKING. That is, you can use just a literal XQuery string here, instead of passing a literal string from SQL to XQuery. In real-world examples, you typically pass a value that is available only at run time.

Example: Updating Multiple Text and Attribute Nodes (Current State)

```

SELECT XMLCast(XMLQuery(' $p/PurchaseOrder/Requestor'
    PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
    AS VARCHAR2(30)) name,
XMLQuery(' $p/PurchaseOrder/LineItems'
    PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
    lineitems
FROM purchaseorder po
WHERE XMLExists(' $p/PurchaseOrder [Reference="SBELL-
    2002100912333601PDT"] ' PASSING po.OBJECT_VALUE AS "p");

```

NAME	LINEITEMS
Sarah J. Bell	<LineItems> <LineItem ItemNumber="1"> <Description>A Night to Remember</Description> <Part Id="715515009058" UnitPrice="39.95" Quantity="2"/> <LineItem> <LineItem ItemNumber="2"> <Description>The Unbearable Lightness Of Being</Description> <Part Id="37429140222" UnitPrice="29.95" Quantity="2"/> <LineItem> <LineItem ItemNumber="3"> <Description>Sisters</Description> <Part Id="715515011020" UnitPrice="29.95" Quantity="4"/> <LineItem> </LineItems>



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the next example, you will learn how to update multiple text and attribute nodes in an XML document. Before the update operations are performed, this SELECT query is used to display the current state of the highlighted XML document.

Example: Updating Multiple Text and Attribute Nodes (Update Operation)

```
UPDATE purchaseorder
SET OBJECT_VALUE =
  XMLQuery('copy $i := $p1 modify
    ((for $j in $i/PurchaseOrder/Requestor
      return replace value of node $j with $p2),
     (for $j in $i/PurchaseOrder/LineItems/LineItem[1]/Part/@Id
      return replace value of node $j with $p3),
     (for $j in $i/PurchaseOrder/LineItems/LineItem[1]/Description
      return replace value of node $j with $p4),
     (for $j in $i/PurchaseOrder/LineItems/LineItem[3]
      return replace node $j with $p5))
    return $i'
  PASSING OBJECT_VALUE AS "p1",'Stephen G. King' AS "p2",
        '786936150421' AS "p3",'The Rock' AS "p4",
        XMLType('<LineItem ItemNumber="99">
          <Description>Dead Ringers</Description>
          <Part Id="715515009249" UnitPrice="39.95"
            Quantity="2"/>
        </LineItem>') AS "p5"
  RETURNING CONTENT)
WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]' '
  PASSING OBJECT_VALUE AS "p");
```

1 row updated.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example updates multiple text and attribute nodes (the blue text) for the highlighted XML document.

Example: Updating Multiple Text and Attribute Nodes (Result)

```
SELECT XMLCast(XMLQuery('$p/PurchaseOrder/Requestor'
    PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
AS VARCHAR2(30)) name,
XMLQuery('$p/PurchaseOrder/LineItems'
    PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT) lineitems
FROM purchaseorder po
WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-
2002100912333601PDT"]' PASSING po.OBJECT_VALUE AS "p");
```

NAME	LINEITEMS
Stephen G. King	<LineItems> <LineItem ItemNumber="1"> <Description>The Rock</Description> <Part Id="786936150421" UnitPrice="39.95" Quantity="2"/> <LineItem> <LineItem ItemNumber="2"> <Description>The Unbearable Lightness Of Being </Description> <Part Id="37429140222" UnitPrice="29.95" Quantity="2"/> <LineItem> <LineItem ItemNumber="99"> <Description>Dead Ringers</Description> <Part Id="715515009249" UnitPrice="39.95" Quantity="2"/> <LineItem> </LineItems>



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If the SELECT query that we used earlier is reissued, the result of the update operations from the last slide is displayed in the slide.

Example: Updating Selected Nodes Within a Collection (Current State)

```
ROLLBACK;

SELECT XMLCast(XMLQuery('$p/PurchaseOrder/Requestor'
    PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
    AS VARCHAR2(30)) name, XMLQuery('$p/PurchaseOrder/LineItems'
    PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
    lineitems
FROM purchaseorder po
WHERE XMLExists('$.PurchaseOrder[Reference="SBELL-
    2002100912333601PDT"]' PASSING po.OBJECT_VALUE AS "p");
```

```
NAME
-----
LINEITEMS
-----
Sarah J. Bell
<LineItems>
  <LineItem ItemNumber="1">
    <Description>A Night to Remember</Description>
    <Part Id="715515009058" UnitPrice="39.95" Quantity="2"/>
  </LineItem>
  <LineItem ItemNumber="2">
    <Description>The Unbearable Lightness Of Being</Description>
    <Part Id="3742914022" UnitPrice="29.95" Quantity="2"/>
  </LineItem>
  <LineItem ItemNumber="3">
    <Description>Sisters</Description>
    <Part Id="715515011020" UnitPrice="29.95" Quantity="4"/>
  </LineItem>
</LineItems>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example updates selected nodes within a collection in an XML document. Before we issue the UPDATE operation, we use the SELECT query to display the partial current status of the highlighted XML document.

Example: Updating Selected Nodes Within a Collection (Update Operation)

```
UPDATE purchaseorder
SET OBJECT_VALUE = XMLQuery('copy $i := $p1 modify
    ((for $j in $i/PurchaseOrder/Requestor
        return replace value of node $j with $p2),
     (for $j in
         $i/PurchaseOrder/LineItems/LineItem/Part[@Id="715515009058"]/@Quantity
        return replace value of node $j with $p3),
     (for $j in $i/PurchaseOrder/LineItems/LineItem
        [Description/text()="The Unbearable Lightness Of Being"]
        return replace node $j with $p4))
    return $i'
PASSED OBJECT_VALUE AS "p1",'Stephen G. King' AS "p2",
25 AS "p3", XMLType('<LineItem ItemNumber="99">
<Part Id="786936150421" Quantity="5" UnitPrice="29.95"/>
<Description>The Rock</Description> </LineItem>') AS "p4"
RETURNING CONTENT)
WHERE XMLExists('$p/PurchaseOrder[Reference=
"SBELL-2002100912333601PDT"]'
PASSED OBJECT_VALUE AS "p");
```

1 row updated.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example updates the selected nodes within a collection for the highlighted XML document.

Example: Updating Selected Nodes Within a Collection (Result)

Requestor Name Sara J. Bell is replaced with Stephen G. King.

Quantity attribute value 2 is replaced with value 25.

```

NAME          LINEITEMS
-----
Stephen G. King <LineItems>
  <LineItem ItemNumber="1">
    <Description>A Night to Remember</Description>
    <Part Id="715515009058" UnitPrice="39.95" Quantity="25"/>
  </LineItem>
  <LineItem ItemNumber="99">
    <Part Id="786936150421" Quantity="5" UnitPrice="29.95"/>
    <Description>The Rock</Description>
  </LineItem>
  <LineItem ItemNumber="3">
    <Description>Sisters</Description>
    <Part Id="715515011020" UnitPrice="29.95" Quantity="4"/>
  </LineItem>
</LineItems>

```

Annotations:

- A red arrow points from the text "Requestor Name Sara J. Bell is replaced with Stephen G. King." to the name "Stephen G. King" in the XML output.
- A red arrow points from the text "Quantity attribute value 2 is replaced with value 25." to the quantity attribute "Quantity="25"" in the XML output.
- A red bracket on the left side of the XML output indicates the replacement of ItemNumber 2 with ItemNumber 99, spanning from the first LineItem to the second LineItem.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The result of the previous slide example is shown here. The result shows the following changes:

- The Requestor name, Sara J. Bell, is replaced with Stephen G. King.
- The Quantity attribute value for ItemNumber 1 is changed from 2 to 25.
- ItemNumber 2 is replaced with ItemNumber 99.

Note: The screen capture is the result of running code_09_17_s.sql.

Updating XML Data to NULL Values Considerations

The following considerations apply to updating XML data to NULL values:

- If you update an XML element to NULL:
 - The attributes and children of the element are removed.
 - The element becomes empty.
 - The type and namespace properties of the element are retained.
- If you update an attribute value to NULL, the value appears as the empty string.
- If you update the text node of an element to NULL, the content (text) of the element is removed.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The following considerations apply to updating XML data to NULL values.

If you update an XML element to NULL, the attributes and children of the element are removed and the element becomes empty. The type and namespace properties of the element are retained.

If you update an attribute value to NULL, the value appears as the empty string.

If you update the text node of an element to NULL, the content (text) of the element is removed. The element itself remains, but it is empty.

We will go over some examples on updating XML data to NULL values on the next three slides.

Example: Updating XML Data to NULL Values (Current State)

```
SELECT XMLCast(XMLQuery('$/PurchaseOrder/Requestor'
    PASSING po.OBJECT_VALUE AS "p"
    RETURNING CONTENT)
    AS VARCHAR2(30)) name,
XMLQuery('$/PurchaseOrder/LineItems'
    PASSING po.OBJECT_VALUE AS "p" RETURNING
    CONTENT) lineitems
FROM purchaseorder po
WHERE XMLExists('$/PurchaseOrder[Reference="SBELL-
    2002100912333601PDT"]' PASSING po.OBJECT_VALUE AS "p");
```

```
NAME
-----
LINEITEMS
-----
Sarah J. Bell
<LineItems>
  <LineItem ItemNumber="1">
    <Description>A Night to Remember</Description>
    <Part Id="715515009058" UnitPrice="39.95" Quantity="2"/>
  </LineItem>
  <LineItem ItemNumber="2">
    <Description>The Unbearable Lightness Of Being</Description>
    <Part Id="37429140222" UnitPrice="29.95" Quantity="2"/>
  </LineItem>
  <LineItem ItemNumber="3">
    <Description>Sisters</Description>
    <Part Id="715515011020" UnitPrice="29.95" Quantity="4"/>
  </LineItem>
</LineItems>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the next example, you will learn how to update XML Data to NULL values. Before updating the XML document, the SELECT query is used to display the content of the highlighted XML document (highlighted in red).

Example: Updating XML Data to NULL Values (Update Operation)

```
UPDATE purchaseorder
SET OBJECT_VALUE =
XMLQuery(
'copy $i := $p1 modify
((for $j in
  $i/PurchaseOrder/LineItems/LineItem[Part/@Id="715515009058"]/
  Description return replace value of node $j with ())
, 
(for $j in $i/PurchaseOrder/LineItems/LineItem/Part[@Id="715515009058"]/
  @Quantity return replace value of node $j with ())
,
(for $j in $i/PurchaseOrder/LineItems/LineItem
[Description/text()= "The Unbearable Lightness Of Being"]
return replace node $j with $p2)
return $i'
PASSING OBJECT_VALUE AS "p1", NULL AS "p2"
RETURNING CONTENT)
WHERE XMLExists('$/PurchaseOrder[Reference="SBELL-
  2002100912333601PDT"]'
PASSING OBJECT_VALUE AS "p") ;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example updates all of the following elements to NULL:

- The `Description` element and the `Quantity` attribute of the `LineItem` element whose `Part` element has attribute `Id` value `715515009058`
- The `LineItem` element whose `Description` element has the content (text) `"The Unbearable Lightness Of Being"`

Example: Updating XML Data to NULL Values (Result)

The screenshot shows the execution of an XML query in Oracle SQL Developer. The query is as follows:

```

SELECT XMLCast(XMLQuery('$p/PurchaseOrder/Requestor'
    PASSING po.OBJECT_VALUE AS "p"
    RETURNING CONTENT)
    AS VARCHAR2(30)) name,
    XMLQuery('$p/PurchaseOrder/LineItems'
        PASSING po.OBJECT_VALUE AS "p" RETURNING
        CONTENT) lineitems
FROM purchaseorder po
WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
    PASSING po.OBJECT_VALUE AS "p");
ROLLBACK;

```

The output in the 'Script Output' window shows the results:

NAME	LINEITEMS
Sarah J. Bell	<LineItems> <LineItem ItemNumber="1"> <Description/> <Part Id="715515009058" UnitPrice="39.95" Quantity="" /> </LineItem> <LineItem ItemNumber="3"> <Description>Sisters</Description> <Part Id="715515011020" UnitPrice="29.95" Quantity="4" /> </LineItem> </LineItems>

rollback complete.

- The **Description** element and the **Quantity** attribute are set to **NULL**.
- The second **LineItem** with attribute **ItemNumber = "2"** becomes empty.

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The previous slide example updated to NULL the **Description** element and the **Quantity** attribute of the **LineItem** element whose **Part** element has attribute **Id** value 715515009058 (the first **LineItem**). Therefore, the output here shows the empty **Description** element and **Quantity** attribute.

The previous slide example also updated to NULL the **LineItem** element whose **Description** element has the content (text) "The Unbearable Lightness Of Being". As a result, the second **LineItem** element is empty.

Inserting Child XML Nodes

- You can use XQuery Update to insert new children under parent XML elements.
- The XML document that is the target of the insertion can be:
 - Schema-based, or
 - Non-schema-based



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use XQuery Update to insert new children, either as a single attribute or as one or more elements of the same type, under parent XML elements. The XML document that is the target of the insertion can be schema-based or non-schema-based.

In the next few slides, some examples of inserting child XML nodes have been provided.

Example: Inserting an Element into a Collection (Current State)

```
SELECT
    XMLQuery(' $p/PurchaseOrder/LineItems/LineItem[@ItemNumber=2
    22]''
    PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
FROM purchaseorder po
WHERE XMLExists(' $p/PurchaseOrder[Reference="AMCEWEN-
    20021009123336171PDT"]''
    PASSING po.OBJECT_VALUE AS "p") ;
```

```
SQL> SELECT XMLQuery(' $p/PurchaseOrder/LineItems/LineItem[@ItemNumber=222]''
    PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
FROM purchaseorder po
WHERE XMLExists(' $p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]''
    PASSING po.OBJECT_VALUE AS "p"); 2 3 4 5
XMLQUERY(' $p/PURCHASEORDER/LINEITEMS/LINEITEM[@ITEMNUMBER=222]' PASSING po.OBJECT_
-----
SQL> ■
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the next slide example, you will learn how to insert an element into a collection. Before performing the insert operation, the highlighted XML document is queried. There is no LineItem where the ItemNumber is 222.

Example: Inserting an Element into a Collection (Insert Operation)

```
UPDATE purchaseorder
SET OBJECT_VALUE =
  XMLQuery('copy $i := $p1 modify
(for $j in $i/PurchaseOrder/LineItems
return (# ora:child-element-name LineItem #)
{insert node $p2 into $j})
return $i'
PASSING OBJECT_VALUE AS "p1",
XMLType('<LineItem ItemNumber="222">
<Description>The Harder They Come</Description>
<Part Id="953562951413" UnitPrice="22.95" Quantity="1"/>
</LineItem>') AS "p2"
RETURNING CONTENT)
WHERE XMLExists('$p/PurchaseOrder [Reference="AMCEWEN-
  20021009123336171PDT"]'
PASSING OBJECT_VALUE AS "p");
```



The Oracle logo, consisting of the word 'ORACLE' in a red sans-serif font.

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example inserts a new `LineItem` element as a child of element `LineItems`. The example uses the Oracle XQuery pragma `ora:child-element-name` to specify the name of the inserted child element as `LineItem`.

Example: Inserting an Element into a Collection (Result)

```
SELECT
  XMLQuery(' $p/PurchaseOrder/LineItems/LineItem[@ItemNumber=
  222]''
  PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
FROM purchaseorder po
WHERE XMLExists(' $p/PurchaseOrder [Reference="AMCEWEN-
  20021009123336171PDT"]''
  PASSING po.OBJECT_VALUE AS "p") ;
```

The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the XQuery code used to insert a new LineItem element with ItemNumber 222. The output shows the resulting XML document structure, which includes a single `<LineItem ItemNumber="222">` element containing a `<Description>` and a `<Part Id="953562951413" UnitPrice="22.95" Quantity="1"/>`.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

After executing the XQuery Update operation in the previous example, the XML document is queried to ensure that the insert operation was successful.

If the XML data to be updated is XML Schema-based and it refers to a namespace, the data to be inserted must also refer to the same namespace. Otherwise, an error is raised because the inserted data does not conform to the XML Schema.

The result in the slide shows that there is only one LineItem where the ItemNumber is 222.

Example: Inserting an Element Before an Element (Current State)

```
SELECT XMLQuery('$/PurchaseOrder/LineItems/LineItem[1] '
PASSED po.OBJECT_VALUE AS "p" RETURNING CONTENT)
FROM purchaseorder po
WHERE XMLExists('$/PurchaseOrder [Reference="AMCEWEN-
    20021009123336171PDT"] '
PASSED po.OBJECT_VALUE AS "p");
```

```
XMLQUERY ('$P/PURCHASEORDER/LINEITEMS/LINEITEM[1]' PASSINGPO.OBJECT_
-----
<LineItem ItemNumber="1">
    <Description>Salesman</Description>
    <Part Id="37429158920" UnitPrice="39.95" Quantity="2" />
</LineItem>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the next slide example, you will learn how to insert an element before another element. Before the insert operation is performed, the highlighted XML document is queried. The query returns the `LineItem` element which has the `ItemNumber` attribute value of 1.

Example: Inserting an Element Before an Element (Result)

```
UPDATE purchaseorder
SET OBJECT_VALUE =
XMLQuery('copy $i := $p1 modify
(for $j in $i/PurchaseOrder/LineItems/LineItem[1]
return insert node $p2 before $j)
return $i'
PASSING OBJECT_VALUE AS "p1",
XMLType('<LineItem ItemNumber="314">
<Description>Brazil</Description>
<Part Id="314159265359" UnitPrice="69.95"
Quantity="2"/>
</LineItem>') AS "p2"
RETURNING CONTENT)
WHERE XMLExists('$p/PurchaseOrder [Reference="AMCEWEN-
20021009123336171PDT"] '
PASSING OBJECT_VALUE AS "p");
```

1 row updated.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To view the results, issue the following query:

```
SELECT
  XMLQuery('$p/PurchaseOrder/LineItems/LineItem[position()<=2] '
PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
FROM purchaseorder po
WHERE XMLExists('$p/PurchaseOrder [Reference="AMCEWEN-
20021009123336171PDT"] '
PASSING po.OBJECT_VALUE AS "p");
```

The following is the result of the update operation in the slide. The new LineItem with ItemNumber 314 is added before the LineItem with ItemNumber.

```
XMLQUERY('$P/PURCHASEORDER/LINEITEMS/LINEITEM[POSITION()<=2]' PASSING PO.OBJECT_VALUE AS "p")
-----  

<LineItem ItemNumber="314"><Description>Brazil</Description><Part Id="314159265359" UnitPrice="69.95" Quantity="2"/></LineItem><LineItem ItemNumber="1"><Description>Salesman</Description><Part Id="37429158920" UnitPrice="39.95" Quantity="2" /></LineItem>
```

Example: Inserting an Element as the Last Child Element (Current State)

```
SELECT XMLQuery('$/PurchaseOrder/Actions/Action[1]'  
PASSED po.OBJECT_VALUE AS "p" RETURNING CONTENT)  
FROM purchaseorder po  
WHERE XMLExists('$/PurchaseOrder [Reference= "AMCEWEN-'  
    20021009123336171PDT"]'  
PASSED po.OBJECT_VALUE AS "p");
```

```
XMLQUERY('$/PURCHASEORDER/ACTIONS/ACTION[1]'PASSEDPO.OBJECT_VALUEAS"P"RETURNIN  
-----  
<Action>  
  <User>KPARTNER</User>  
</Action>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the next slide example, you will learn how to insert an element as the last child element. Before the insert operation is performed, the XML document is queried as shown in the slide.

Example: Inserting an Element as the Last Child Element (Update and Result)

```
UPDATE purchaseorder
  SET OBJECT_VALUE =
    XMLQuery('copy $i := $p1 modify
              (for $j in $i/PurchaseOrder/Actions/Action[1]
               return insert nodes $p2 as last into $j)
              return $i'
              PASSING OBJECT_VALUE AS "p1",
                    XMLType('<Date>2002-11-04</Date>') AS "p2"
              RETURNING CONTENT)
 WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-
 20021009123336171PDT"]'
 PASSING OBJECT_VALUE AS "p");
```

1 row updated.

```
XMLQUERY('$P/PURCHASEORDER/ACTIONS/ACTION[1]' PASSING po.OBJECT_VALUE AS "P" RETURNIN
-----  
<Action>
  <User>KPARTNER</User>
  <Date>2002-11-04</Date>
</Action>
```

SQL> ■

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the slide example, insert the Date element is inserted before the user element. If you reissue the query from the previous slide, the result shows the newly added Date element.

```
SELECT XMLQuery(' $p/PurchaseOrder/Actions/Action[1] '
  PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
  FROM purchaseorder po
 WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-
 20021009123336171PDT"]'
  PASSING po.OBJECT_VALUE AS "p");
```

Example: Deleting XML Nodes (Current State)

```
SELECT
  XMLQuery(' $p/PurchaseOrder/LineItems/LineItem[@ItemNumber=23]' .
    PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
FROM purchaseorder po
WHERE XMLExists(' $p/PurchaseOrder [Reference="AMCEWEN-
  20021009123336171PDT"]' .
  PASSING po.OBJECT_VALUE AS "p");
```

```
XMLQUERY(' $P/PURCHASEORDER/LINEITEMS/LINEITEM[@ITEMNUMBER=23]' PASSING PO.OBJECT_V
-----
<LineItem ItemNumber="23">
  <Description>Great Expectations</Description>
  <Part Id="37429128022" UnitPrice="39.95" Quantity="4"/>
</LineItem>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the next slide example, you will learn how to delete XML nodes. Before performing the delete operation, the highlighted XML document is queried.

Example: Deleting XML Nodes (Result)

```
UPDATE purchaseorder
SET OBJECT_VALUE =
XMLQuery('copy $i := $p modify
          delete nodes
          $i/PurchaseOrder/LineItems/LineItem[@ItemNumber="23"]
          return $i'
          PASSING OBJECT_VALUE AS "p" RETURNING CONTENT)
WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-
  20021009123336171PDT"]'
          PASSING OBJECT_VALUE AS "p");
```

1 row updated.

```
XMLQUERY('$P/PURCHASEORDER/LINEITEMS/LINEITEM[@ITEMNUMBER=23]'PASSINGPO.OBJECT_V
-----
```

SQL> ■



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example deletes the LineItem element whose ItemNumber attribute has the value 23 where the Reference node is "AMCEWEN-20021009123336171PDT". If you reissue the query from the previous slide, no rows are selected.

Example: Creating XML Views of Modified XML Data

```
CREATE OR REPLACE VIEW purchaseorder_summary OF XMLType AS
  SELECT XMLQuery('copy $i := $p1 modify
    ((for $j in $i/PurchaseOrder/Actions
      return replace value of node $j with ()),
     (for $j in $i/PurchaseOrder/ShippingInstructions
      return replace value of node $j with ()),
     (for $j in $i/PurchaseOrder/LineItems
      return replace value of node $j with ()))
    return $i'
  PASSING OBJECT_VALUE AS "p1" RETURNING CONTENT)
FROM purchaseorder p;
```

View created.

SQL> ■



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use XQuery Update to create new views of XML data. The slide example creates a new view of the `purchaseorder` table named `purchaseorder_summary` where we are setting the `Actions`, `ShippingInstructions`, and `LineItems` nodes to NULL values.

Example: Creating XML Views of Modified XML Data (Result)

```
SELECT OBJECT_VALUE FROM purchaseorder_summary  
WHERE XMLExists('$/PurchaseOrder[Reference="DAUSTIN-  
20021009123335811PDT"]'  
PASING OBJECT_VALUE AS "p");
```

OBJECT_VALUE

```
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames  
paceSchemaLocation="http://localhost:8080/source/schemas/poSource/xsd/purchaseOr  
der.xsd"><Reference>DAUSTIN-20021009123335811PDT</Reference><Actions></Actions><  
Reject/><Requester>David L. Austin</Requester><Requestor>User=DAUSTIN</User><CostCenter>S3  
0</CostCenter><ShippingInstructions></ShippingInstructions><SpecialInstructions>  
Courier</SpecialInstructions><LineItems></LineItems></PurchaseOrder>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The SELECT query in the slide displays the newly created purchaseorder_summary view.

Using XQuery Update to Transform Data

```
SQL>select * from employee;  
  
SYS_NC_ROWINFO$  
-----  
<employees>  
<employee>  
<name>John Doe</name>  
<SSN>123-45-6789</SSN>  
</employee>  
</employees>
```

```
select xmlquery('copy $e := $emp modify for $i in $e/employees  
return delete node $i//SSN return $e'  
passing object_value as "emp" returning content)  
from employee;
```

```
XMLQUERY('COPY$E:=EMPLOYEE/EMPLOYEES/EMPLOYEE/SSNRE  
-----  
<employees>  
<employee>  
<name>John Doe</name>  
</employee>  
</employees>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use XQuery Update to transform XML data. The first code example assumes that there is an employee table that contains one XML employee record. This record contains among other elements a social security element which you do not want to show. You can hide an element by using the delete node syntax as shown in the second code example. The result of the second code example is shown in the slide.

The code to create and populate the employee table are as follows:

```
CREATE TABLE employee of XMLType;  
INSERT INTO employee  
VALUES (XMLType ('<employees>  
    <employee>  
        <name>John Doe</name>  
        <SSN>123-45-6789</SSN>  
    </employee>  
</employees>')) ;
```

Quiz

You can use XQuery Update operations to modify the values of existing nodes, replace a fragment of XML with another fragment of XML, and insert and remove nodes from the document.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a

You can use XQuery Update operations to modify the values of existing nodes, replace a fragment of XML with another fragment of XML, and insert and remove nodes from the document. In Oracle XML DB, you execute XQuery operations by using the `XMLQuery` operator.

Summary

In this lesson, you should have learned how to:

- Update an entire XML document
- Replace XML nodes
- Use SQL UPDATE and XQuery Update
- Update XML data to NULL values
- Insert an element into a collection
- Delete XML nodes
- Create a view by using updated XML data



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 9-1: Overview

This practice covers using XQuery update to insert a Date element as the last child of an Action element.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 9-2: Overview

This practice covers using XQuery update to insert an element after another element.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

10

Searching XML Content Using XQuery Full-Text



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Learn about the basic concepts of Oracle Text
- Identify the XQuery full-text search capabilities
- Create the supporting Oracle Text structures required to create a full-text index
- Create a full-text index
- Use XQuery full-text queries that will use the full-text index



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XQuery Full-Text Search Capabilities

- XQuery Full Text adds a full-text contains expression to the XQuery language.
- You use such an expression in your query to search:
 - The text of element nodes and their descendant elements
 - The text of attribute nodes



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XQuery and XPath Full Text 1.0 Recommendation (referred to in this self-study as XQuery Full Text) defines XQuery support for full-text searches in queries. It defines full-text selection operators that perform the search and return instances of the `AllMatches` model, which complements the XQuery Data Model (XDM). An `AllMatches` instance describes all possible solutions for a full-text query for a given search context item. Each solution is described by a `Match` instance, which contains the search-context tokens that must be included (`StringInclude` instances) and excluded (`StringExclude` instances).

XQuery Full Text adds a full-text `contains` expression to the XQuery language. You use such an expression in your query to search the text of element nodes and their descendant elements (you can also search the text of attribute nodes).

Combining Oracle Text Features With Oracle XML DB

- You can combine the features of Oracle Text and Oracle XML DB for applications where you want to do a full-text retrieval, leveraging the XML structure.
- You can enter queries such as "*find all nodes that contain the word Pentium.*"
- Oracle Database 12c extends Oracle's support for the W3C XQuery specification by adding support for the XQuery full-text extension, which enables you to perform XML-aware full text searches on XML content stored in the database.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Comparison of Full-Text Search and Other Search Types

Full-text search differs from structured search or substring search in the following ways:

- A full-text search looks for whole words rather than substrings.
- A full-text search supports some language-based and word-based searches that substring searches do not.
- A full-text search generally involves some notion of relevance.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Full-text search differs from structured search or substring search in the following ways:

- A full-text search looks for whole words rather than substrings. A substring search for comments that contain the string "law" can return a comment that contains "my lawn is going wild". A full-text search for the word "law" cannot.
- A full-text search supports some language-based and word-based searches that substring searches do not. You can use a language-based search, for example, to find all the comments that contain a word with the same linguistic stem as "mouse", and Oracle Text finds "mouse" and "mice". You can use a word-based search, for example, to find all the comments that contain the word "lawn" within 5 words of "wild".
- A full-text search generally involves some notion of relevance. When you do a full-text search for all the comments that contain the word "lawn", for example, some results are more relevant than others. Relevance is often related to the number of times the search word (or similar words) occur in the document.

Indexing for XQuery Full Text

- You can use XQuery Full Text to query full-text content within XML data.
- With XQuery Full Text, you create an appropriate Oracle Text (full-text) index. The index is an XML full-text index, which is designed specifically for use with XML data.



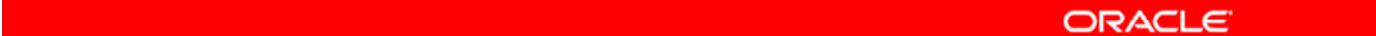
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To query full-text content within XML data, you can use XQuery Full Text (XQFT) or Oracle-specific full-text constructs. In either case, you create an appropriate Oracle Text (full-text) index. In the case of XQFT, the index is an XML full-text index, which is designed specifically for use with XML data. This section covers the creation of an Oracle Text index and its use with XML data.

You can perform XQuery Full Text queries on `XMLType` data. If you use an XQuery Full Text full-text predicate in an `XMLExists` expression within a SQL `WHERE` clause, you must create an XML-enabled Oracle Text index (also referred to as an XML full-text index).

Available Documentation

- *Oracle Text Application Developer's Guide 12c Release 1 (12.1)*
- *Oracle Text Reference 12c Release 1 (12.1)*
- *Oracle XML DB Developer's Guide 12c Release 1 (12.1)*

The Oracle logo, which consists of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

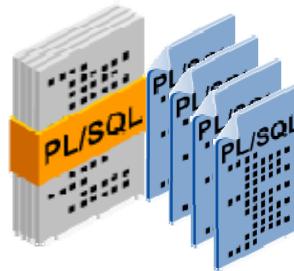
For additional information on XQuery Full Text, see the following documentation guides:

- *Oracle Text Application Developer's Guide 12c Release 1 (12.1)*
- *Oracle Text Reference 12c Release 1 (12.1)*
- *Oracle XML DB Developer's Guide 12c Release 1 (12.1)*

A general rule for understanding Oracle support for XQuery Full Text is that the Oracle implementation is based on Oracle Text. Oracle Text provides full-text indexing and searching capabilities for Oracle products and for applications developed with such products.

The CTX_DDL Package

- Use the CTX_DDL PL/SQL package procedures to create and manage the preferences, section groups, and stoplists required for Text indexes.
- We will use several of this package procedures in this section of the lesson.
- For additional information on the CTX_DDL package procedures, see the Oracle Text Reference 12c Release 1 (12.1) reference guide.



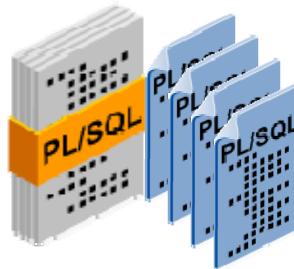
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The CTX_DDL PL/SQL package provides procedures to create and manage the preferences, section groups, and stoplists required for Text indexes.

The CTX_DDL Package Procedures Used in This Lesson

Procedure	Description
create_section_group	Creates a section group for defining sections in a text column.
set_sec_grp_attr	Adds a section group-specific attribute to a section group identified by name.
create_preference	Creates a preference in the Text data dictionary.
set_attribute	Sets a preference attribute.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle Text document section searching enables you to narrow text queries down to blocks of text within documents. Section searching is useful when your documents have internal structure, such as HTML and XML documents. You can also search for text at the sentence and paragraph level.

The CTX_DDL package contains the several stored procedures and functions. The following are some of the procedures and functions used in this lesson:

- CREATE_SECTION_GROUP: Creates a section group for defining sections in a text column.
- SET_SEC_GRP_ATTR: Adds a section group-specific attribute to a section group identified by name. This procedure is also used to set xml_enable to support XML awareness.
- CREATE_PREFERENCE: Creates a preference in the Text data dictionary.
- SET_ATTRIBUTE: Sets a preference attribute. Use this procedure after you have created a preference with CTX_DDL.CREATE_PREFERENCE.

Note: For detailed information on the CTX_DDL package procedures and functions, section groups, section groups attributes, preferences, and other terms, see the Oracle Text Reference 12c Release 1 (12.1) documentation reference.

Oracle Text Concepts: Oracle Text Section Searching

- Oracle Text document section searching enables you to narrow text queries down to blocks of text within documents.
- Section searching is useful when your documents have internal structure, such as XML documents.
- Path section group automatically creates a zone section for each start-tag/end-tag pair in an XML document.
 - The section names derived from XML tags are case-sensitive as in XML.
 - It allows you to do path searching with the `INPATH` and `HASPATH` operators.
- XML Full Text Index is a context index with path section group and its attribute `xml_enable` set to true.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XML Full Text Index

- XML full text index is an “inverted index” where the positions of all words, elements and attributes are recorded in index tables.
- XML full text index uses an Oracle Text CONTEXT index with the section group attribute XML_ENABLE set to TRUE.
- Three index tables are used
 - \$I table (DR\$indexname\$I) stores word, element and attribute information
 - \$D table (DR\$indexname\$D) stores additional attribute information
 - \$E table (DR\$indexname\$R) stores namespace information



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The \$I table contains the following:

- TOKEN_TYPE: This defines whether the token is a content word, element name, or attribute name.
- TOKEN_TEXT: This contains the content word, element name, or attribute name.
- TOKEN_INFO: This records the document(s) in which the token appears in, and the word offset to the start of the token, or the start and end of the element or attribute.

The \$I, \$D, and \$E tables are created by the system when you issue the CREATE INDEX command.

XML Full Text: Example

```
<emp>
  <name type="first">Peter</name>
  <name type="last">York</name>
  <city>New York</city>
</emp>
```

} document1.xml

TOKEN_TYPE	TOKEN_TEXT	TOKEN_INFO
0 (text)	PETER	<1>5
0 (text)	YORK	<1>10, 14
0 (text)	NEW	<1>13
4 (attribute text)	FIRST	<1>4
4 (attribute text)	LAST	<1>9
7 (element)	emp	<1>1..16
7 (element)	city	<1>12..14
7 (element)	name	<1>1..6, 7..11
8 (attribute)	name@type	<1>3..4, 8..9
8 (attribute)	@type	<1>3..4, 8..9

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the slide example, we are assuming that there is only one xml document named document1.xml. The following is the explanation of the three table columns:

Column 1 shows only few of the available token types. There are many other types of tokens that are not relevant for this example.

Column 2 shows the token's text or attribute from document1.xml.

Column 3 shows the position where the token_text was found. <1> refers to the docid or the internal document number. In this example, because there is only one document involved, document1.xml, the docid is 1.

Enabling Oracle Text Section Searching: Overview

Argument Name	Description
PATH_SECTION_GROUP	Use this group type to index XML documents. Behaves like the AUTO_SECTION_GROUP.
attribute_value	Specify the section group attribute value. Specify boolean values as TRUE or FALSE, T or F, YES or NO, Y or N, ON or OFF, or 1 or 0.
xml_enable	Boolean attribute that determines whether a path section group indexes documents in an XML aware manner. When <code>xml_enable</code> is turned on, then the path section group supports XML awareness. The <code>\$D</code> table is created automatically under an <code>xml_enable</code> Text index. The default setting of <code>xml_enable</code> is FALSE.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Introduction to Section Groups

One of the choices you make when creating a CONTEXT index is section group. A section group instance is based on a section group type. The section group type specifies the kind of structure in your documents, and how to index (and therefore search) that structure. The section group instance may specify which structure elements are indexed. Most users either take the default section group or use a predefined section group.

Choosing a Section Group Type

There are several section group types. The ones that we will use in this lesson are the section group types that are useful in XML searching:

`PATH_SECTION_GROUP`

Choose this when you want to use `WITHIN`, `INPATH` and `HASPATH` in queries, and you want to be able to consider all sections to scope the query.

`XML_SECTION_GROUP`

Choose this when you want to use `WITHIN`, but not `INPATH` and `HASPATH`, in queries, and you want to be able to consider only explicitly-defined sections to scope the query.

`XML_SECTION_GROUP` section group type supports `FIELD` sections in addition to `ZONE` sections. In some cases, `FIELD` sections offer significantly better query performance.

Full-Text contains Expression

A full-text contains expression is an XQuery expression that represents a full-text search.

- A full-text contains expression (`FTContainsExpr`) supported by Oracle has two parts:
 - A search context
 - A full-text selection
- The selection part is itself composed of the following:
 - Tokens and phrases used for matching
 - Optional match options, such as stemming
 - Optional Boolean operators for combining full-text selections
 - Optional constraint operators, such as positional filters



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XQuery expressions are case-sensitive. An XQuery expression is either a simple expression or an updating expression, the latter being an expression that represents data modification. In addition to the usual possible XQuery expressions, starting with Oracle Database 12c, the following XQuery expression is possible:

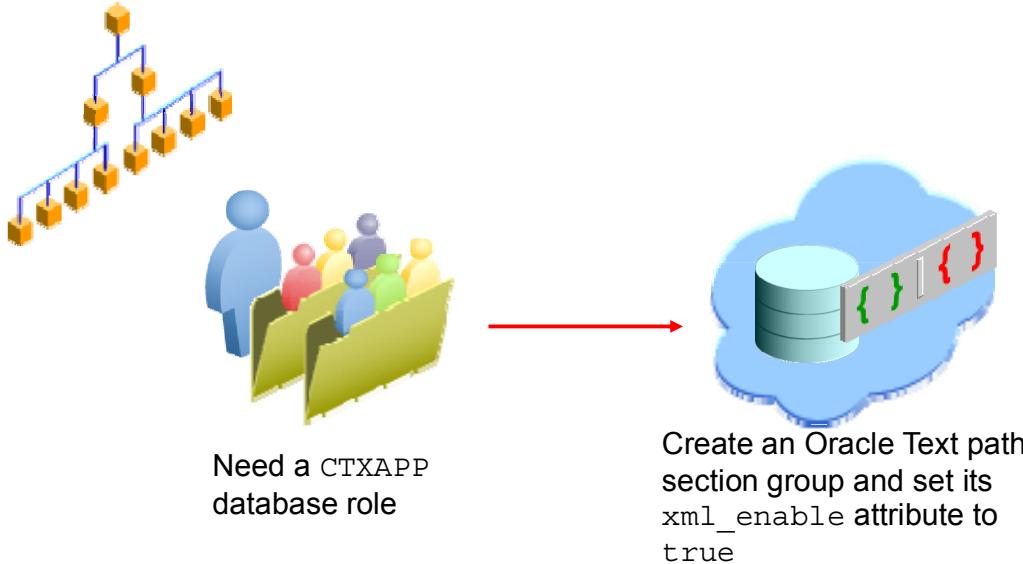
The full-text contains expression is an XQuery expression that represents a full-text search. This expression is provided by the XQuery and XPath Full Text 1.0 Recommendation. A full-text contains expression (`FTContainsExpr`) supported by Oracle has these parts: A search context that specifies the items to search and a full-text selection that filters those items and selects matches.

The selection part is itself composed of the following:

- Tokens and phrases used for matching
- Optional match options, such as the use of stemming
- Optional Boolean operators for combining full-text selections
- Optional constraint operators, such as positional filters (for example, ordered window)

For additional information on XQuery Full Text, see the *Oracle XML DB Developer's Guide 12c Release 1 (12.1)* documentation guide.

Requirements for Creating XQuery Full Text Index

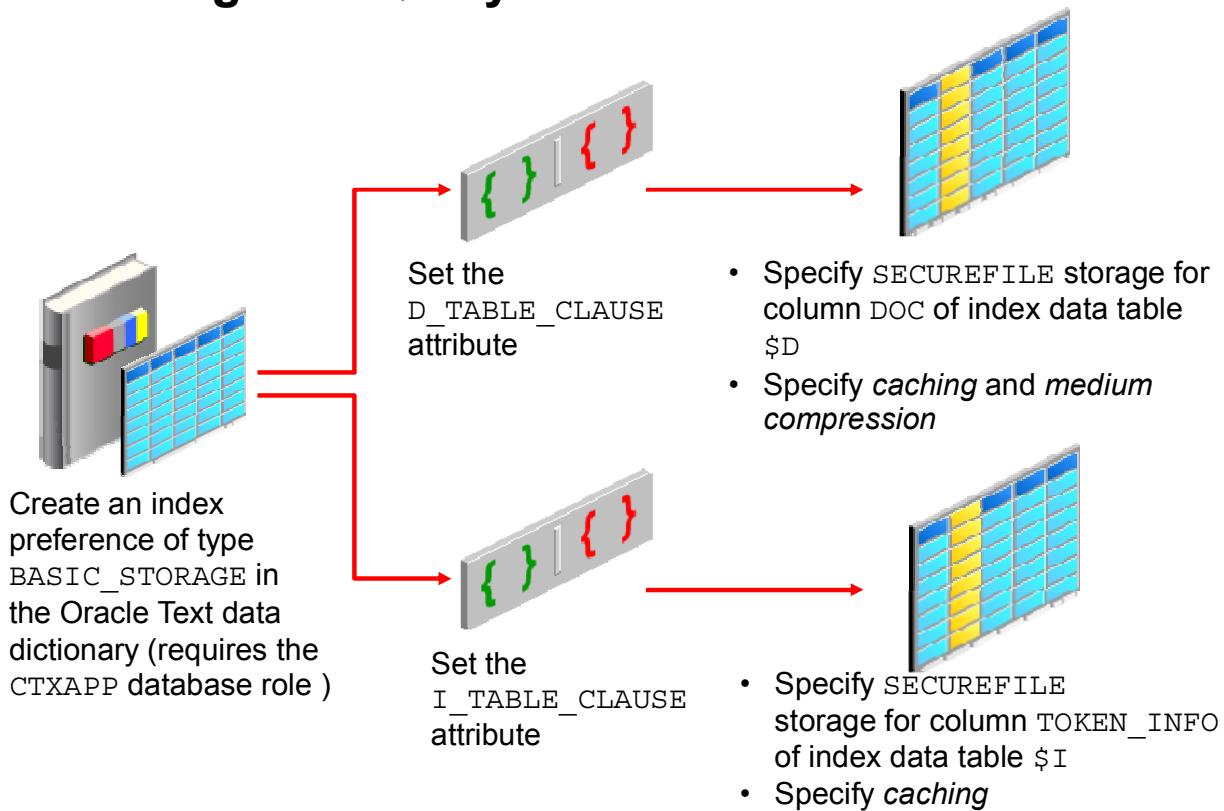


ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To create an XML-enabled Oracle Text index, you must be granted the `CTXAPP` database role. This role is needed to create Oracle Text indexes so that you can set Oracle Text index preferences or use Oracle Text PL/SQL packages. In addition, before you create the index, you must create an Oracle Text path section group and set its `xml_enable` attribute to `true` to make the path section group XML-aware. This is discussed in detail on the next slide.

Indexing for XQuery Full Text: Best Performance



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For best performance, create an index preference of type `BASIC_STORAGE` in the Oracle Text data dictionary, specifying the following attributes:

- **`D_TABLE_CLAUSE`:** Specify `SECUREFILE` storage for column `DOC` of index data table `$D`, which contains information about the structure of your XML documents. Specify caching and medium compression (if you have the license for the compression option).
- **`I_TABLE_CLAUSE`:** Specify `SECUREFILE` storage for column `TOKEN_INFO` of index data table `$I`, which contains information about full-text tokens and their occurrences in the indexed documents. Specify caching but not compression.

Using a Full-Text Index: Syntax and Example

```
-- Syntax: Creates a full-text index on the named XMLType  
-- column of the named table. The index type is ctxsys.context  
  
CREATE INDEX [schema.]index  
ON [schema.]table (XMLType-column)  
INDEXTYPE IS ctxsys.context;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Example: Creating an XML Full-Text Index

```
BEGIN
  CTX_DDL.create_section_group('mysecgroup',
                               'PATH_SECTION_GROUP');
  CTX_DDL.set_sec_grp_attr('mysecgroup', 'XML_ENABLE', 'TRUE');
  CTX_DDL.create_preference('mypref', 'BASIC_STORAGE');
  CTX_DDL.set_attribute('mypref', 'D_TABLE_CLAUSE', 'TABLESPACE
    my_ts LOB(DOC) STORE AS SECUREFILE
      (TABLESPACE my_ts COMPRESS MEDIUM CACHE)');
  CTX_DDL.set_attribute('mypref', 'I_TABLE_CLAUSE',
    'TABLESPACE my_ts LOB(TOKEN_INFO) STORE AS SECUREFILE
      (TABLESPACE my_ts NOCOMPRESS CACHE)');
END;
/
CREATE INDEX po_ctx_idx ON po_binxml(OBJECT_VALUE)
INDEXTYPE IS CTXSYS.CONTEXT
PARAMETERS('storage mypref section group mysecgroup');
```

Creates a full-text index on the po_binxml table. The index type is ctxsys.context



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide examples use a non-XML-schema-based XMLType table named `po_binxml`. This table has the same data as the `purchaseorder` table in the standard OE schema.

The `BASIC_STORAGE` index preference specifies the tablespace and creation parameters for the database tables and indexes that constitute an Oracle Text index.

You have many choices available when building a full-text index. These choices are expressed as indexing preferences. To use an indexing preference, add the `PARAMETERS` clause to `CREATE INDEX`.

Note: The steps in the code example before the `CREATE INDEX` statements are all required in order to create a full-text index.

Example: XQuery Full Text Query

```

SELECT XMLQuery('for $i in /PurchaseOrder/LineItems/LineItem/Description
    where $i[. contains text "Big" ftand "Street"]
    return <Title>{$i}</Title>'
    PASSING OBJECT_VALUE RETURNING CONTENT)
FROM po_binxml
WHERE XMLExists('/PurchaseOrder/LineItems/LineItem/Description
[. contains text "Big" ftand "Street"]'
PASING OBJECT_VALUE);

```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	2014	4 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	PO_BINXML	1	2014	4 (0)	00:00:01
*	DOMAIN INDEX	PO_CTX_IDX			4 (0)	00:00:01

Index po_ctx_idx is used

Predicate Information (identified by operation id):

```

2 - access("CTXSYS"."CONTAINS"(SYS_MAKEXML(0,"XMLDATA"),'<query><textquery
grammar="CONTEXT" lang="english"> ( ( {Big} ) and ( {Street} ) )  INPATH
(/PurchaseOrder/LineItems/LineItem/Description)</textquery></query>')>0)

```

Note

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example queries the data to retrieve the `Description` elements whose text contains both `Big` and `Street`, in that order.

The partial execution plan for the query indicates that index `po_ctx_idx` is used.

As mentioned earlier, if you use an XQuery full-text predicate in an `XMLExists` expression within a SQL WHERE clause, but you do not create an XML-enabled Oracle Text index or the index cannot be used for some reason, compile-time error ORA-18177 is raised.

Oracle Text Reference Documentation Guide Related Topics

See the Oracle Text Reference for information about the following:

- Section groups
- Procedure `CTX_DLL.set_sec_grp_attr`
- Procedure `CTX_DLL.create_preference`
- Procedure `CTX_DLL.set_attribute`
- Preference `BASIC_STORAGE`, `D_TABLE_CLAUSE`, and `I_TABLE_CLAUSE`



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Using XML Schema-Based Data with XQuery Full Text

- Oracle recommends that you use non-XML schema-based XMLType data stored as binary XML when you use XQuery Full Text and an XML full-text index.
- By default, when an XML full-text index is used to evaluate XML schema-based data, the ORA-18177 compile-time error is raised.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle recommends that you use non-XML Schema-based XMLType data stored as binary XML when you use XQuery Full Text and an XML full-text index.

By default, when an XML full-text index is used to evaluate XML Schema-based data, the ORA-18177 compile-time error is raised. Full-text indexing itself makes no use of the associated XML Schema; it is not type-aware. Full-text indexing treats all of the text that it applies to as untyped. Finally, this error is raised even if you type-cast data appropriately and thus do not depend on the XML Schema to cast types implicitly.

Error ORA-18177: Using XML Schema-Based Data with XQuery Full Text

```
SELECT XMLQuery('/PurchaseOrder/LineItems/LineItem'
    PASSING OBJECT_VALUE RETURNING CONTENT)
  FROM oe.purchaseorder
 WHERE XMLExists('/PurchaseOrder'
    [LineItems/LineItem/@ItemNumber > xs:integer("20")
     and Actions/Action/User contains text "KPARTNER"]'
    PASSING OBJECT_VALUE);
  FROM oe.purchaseorder
  *
ERROR at line 3:
ORA-18177: XQuery full text expression '/PurchaseOrder
[LineItems/LineItem/@ItemNumber > xs:integer("20")
and Actions/Action/User contains text "KPARTNER"]' cannot be evaluated using XML
text index
```



Compile error ORA-18177

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The screen capture shows an XQuery Full Text query with XML Schema-based data, which raises the ORA-18177 compile error.

Using XQuery Pragma `ora:no_schema` with XML Schema-Based Data

- You can use XQuery Full Text to query XMLType data regardless of the storage model.
- If you use it with XML Schema-based data, you must also use the XQuery `ora:no_schema` extension-expression pragma in your query; otherwise, an error is raised.
- If you use `ora:no_schema`, the XML data is implicitly cast to non-XML Schema-based data.



Pragma `ora:use_xmltext_idx`

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use XQuery Full Text to query XMLType data regardless of the storage model. However, if you use it with XML Schema-based data, you must also use the XQuery `ora:no_schema` extension-expression pragma in your query; otherwise, an error is raised.

If you use `ora:no_schema` for XQuery Full Text, the XML data is implicitly cast to non-XML Schema-based data. In other words, Oracle support of XQuery Full Text treats all XML data as if it were not based on an XML Schema.

If you include an XQuery Full Text condition that makes use of XML Schema data types in your query, such type considerations are ignored. A comparison of two XML Schema date values, for instance, is handled as a simple string comparison. Oracle support for XQuery Full Text is not-XML Schema-aware.

Example: Using XQuery Pragma ora:no_schema

```
SELECT XMLQuery('/PurchaseOrder/LineItems/LineItem'
                  PASSING OBJECT_VALUE RETURNING CONTENT)
FROM oe.purchaseorder
WHERE XMLExists('(# ora:no_schema #)
                  {/PurchaseOrder
                  [LineItems/LineItem/@ItemNumber >
                   xs:integer("20"))
                  and Actions/Action/User contains text "KPARTNER"] }'
                  PASSING OBJECT_VALUE);
```



PRAGMA ora:use_xmltext_idx

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example shows a query of XML Schema-based data, with explicit type-casting to ensure that the proper condition is evaluated.

Pragma ora:use_xmltext_idx: Forcing an XML Full-Text Index

When evaluating a query that contains XML data, you can force the use of an existing XML full-text index by using the XQuery pragma `ora:use_xmltext_idx`.

```
SELECT XMLQuery('/PurchaseOrder/LineItems/LineItem'
  PASSING OBJECT_VALUE RETURNING CONTENT)
FROM po_binxml
WHERE XMLExists('/PurchaseOrder/LineItems/LineItem
  [Description contains text "Picnic"]' PASSING OBJECT_VALUE)
AND XMLExists('# ora:use_xmltext_idx #
  {/PurchaseOrder [User="SBELL"]} '
  PASSING OBJECT_VALUE);
```



Pragma ora:use_xmltext_idx

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can evaluate a query involving XML data in various ways, depending on whether different types of indexes are available, in addition to other factors. Sometimes the default evaluation method is not the most efficient, and it would be more efficient to force the use of an existing XML full-text index. You can use XQuery pragma `ora:use_xmltext_idx` to do this.

For example, a WHERE clause includes two `XMLExists` expressions, only one of which involves an XQuery full-text condition. You also have an `XMLIndex` index that applies to the `XMLExists` expression which has no full-text condition. With such a query, it is typically more efficient to use an XML full-text index to evaluate the entire WHERE clause.

Even in some cases where there is no XQuery full-text condition in the query, an XML full-text index can provide the most efficient query evaluation.

The query in the slide example uses the pragma `ora:use_xmltext_idx`. Only the first `XMLExists` clause uses a full-text condition. Because of the pragma, the `po_ctx_idx` full-text index, which was created earlier, is used for both `XMLExists` clauses.

Migrating from Oracle Text Index to XML Full-Text Index for XML

- Starting with Oracle Database 12c Release 1 (12.1), the XQuery and XPath Full Text standard is supported by Oracle XML DB.
- Oracle recommends that you migrate legacy code Oracle-specific constructs with XQuery and XPath Full Text.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Starting with Oracle Database 12c Release 1 (12.1), the XQuery and XPath Full Text standard is supported by Oracle XML DB. Prior to this release, for full-text querying of XML data, you could use only an Oracle Text index that was not XML-enabled. In addition, your full-text queries must use Oracle-specific constructs: SQL function `CONTAINS` or XPath function `ora:contains`. Oracle recommends that you migrate your legacy code to use XQuery and XPath Full Text.

Example: Migrating from Oracle Text Index to XML Full Text Index for XML

```
-- Original example.

CONTAINS(t.x, 'Big INPATH
(/P/LIs/LI/Description)') > 0
```

```
-- Replacement example.

XMLExists('$d/P/LIs/LI/Description
[. contains text "Big"]'
PASSING t.x AS "d")

-- If the data is XML Schema-based:

XMLExists('# ora:no_schema #')
{$d/P/LIs/LI/Description
[. contains text "Big"]}'
PASSING t.x AS "d")
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example provides a mapping from typical queries that use Oracle-specific constructs to queries that use XQuery Full Text.

Note: For additional examples on which XQuery and XPath Full Text constructs you can use to replace the use of `CONTAINS` and `ora:contains` in queries, see the *Oracle XML DB Developer's Guide 12c Release 1 (12.1)* documentation guide.

Restrictions on Using XQuery Full Text with XMLExists

- You can pass only one `XMLType` instance as a SQL expression in the `PASSING` clause of the `XMLExists` SQL/XML function.
- Each of the other, non-`XMLType` SQL expressions in that clause must be:
 - Either a compile-time constant of a SQL built-in data type, or
 - A bind variable that is bound to an instance of such a data type
- If this restriction is not met, the `ORA-18177` compile-time error is raised.



ORA-18177 error

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can pass only one `XMLType` instance as a SQL expression in the `PASSING` clause of the `XMLExists` SQL/XML function. Each of the other, non-`XMLType` SQL expressions in that clause must be either a compile-time constant of a SQL built-in data type or a bind variable that is bound to an instance of such a data type. If this restriction is not met, the `ORA-18177` compile-time error is raised.

Quiz

You can use XQuery Full Text to query XMLType data only when you use the Object-Relational storage model.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

You can use XQuery Full Text to query XMLType data regardless of the storage model.

Summary

In this lesson, you should have learned how to:

- Learn about the basic concepts of Oracle Text.
- Identify the XQuery full-text search capabilities.
- Create the supporting Oracle Text structures required to create a full-text index.
- Create a full-text index.
- Use XQuery full-text queries that will use the full-text index.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

11

Indexing XMLType Data

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Create an XMLIndex index
- Identify the components of XMLIndex
- View XMLIndex information in a data dictionary
- Determine XMLIndex index usage
- Specify the paths for XMLIndex index
- Use optimizer hints to turn off XMLIndex



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This lesson discusses the types and benefits of indexes that you can create on XMLType instances. The lesson focuses on the advantages and the uses of XMLIndex. You learn how to create an XMLIndex index. You also learn how to narrow the focus of indexing.

Lesson Agenda

- Indexing XMLType data
- Using XMLIndex index
- XMLIndex unstructured component
- XMLIndex structured component
- Guidelines



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Overview of Indexing XMLType Data

- You can create indexes on your XML data to focus on particular parts of it that you query often and thus improve performance.
- Database indexes improve performance by providing faster access to table data.
- You can index XML data using XMLIndex.
- You can also use Oracle Text CONTEXT indexes to supplement the use of XMLIndex.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can create indexes on your XML data, to focus on particular parts of it that you query often and thus improve performance.

Database indexes improve performance by providing faster access to table data. The use of indexes is particularly recommended for online transaction processing (OLTP) environments involving few updates.

You can index XML data using XMLIndex. You can also use Oracle Text CONTEXT indexes to supplement the use of XMLIndex.

Indexing XMLType Data

Improve the performance of search operations on the XML data that is stored in the database by using:

- B-tree indexes
- Full-text indexes
- XMLIndex indexes
 - Structured component
 - Unstructured component



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For different storage models, Oracle XML DB supports the creation of a variety of indexes on the XML content. You can create indexes on one or more table columns, or on a functional expression. You can create XMLIndex and Oracle Text indexes on XML data.

When the XMLType table or column is based on structured storage (o-r), conventional B-tree indexes can be created on the underlying SQL types. XMLIndex provides a general, XML-specific index that indexes the internal structure of XML data by using binary XML, unstructured, and hybrid storage.

The structured component of an XMLIndex index replaces function-based indexes. Full-text indexes, also known as Oracle Text indexes, which were covered in the previous lesson, can be created on any XMLType table or column.

Lesson Agenda

- Indexing XMLType data
- Using XMLIndex index
- XMLIndex unstructured component
- XMLIndex structured component
- Guidelines

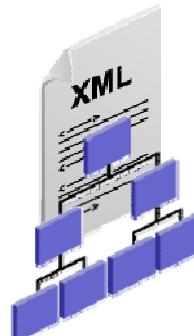


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLIndex: Overview

XMLIndex:

- Provides a general, XML-specific index that indexes the internal structure of XML data
- Overcomes the indexing limitation presented by unstructured and binary XML storage of XML data



XMLIndex

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLIndex is an index, which was introduced in Oracle Database 11g. XMLIndex is a domain index consisting of underlying physical tables and secondary indexes. It provides full support for extremely efficient XPath-based searching of the indexed XML content.

You use B-tree indexes with structured storage because they target the underlying objects directly. But B-tree indexes are not advantageous in addressing the detailed structure (elements and attributes) of an XML document or fragment stored in a CLOB instance. In such special cases of unstructured and binary XML storage, XMLIndex is advantageous. XMLIndex indexes the XML tags of your document and identifies document fragments based on the XPath expressions that target them. XMLIndex also records document hierarchy for each indexed node: relations of parent/child, ancestor/descendant, and siblings.

Following are two examples where usage of XMLIndex index is advantageous:

- You may want to access certain portions of a document in their entirety, so you might pack those portions into one or more CLOB instances. XMLIndex is useful if you want to query within these document portions.
- An XML schema may contain xsd:any elements, for lack of prior information about the document structure and the data types involved. The data in these elements is stored in CLOB instances, and you can use XMLIndex to speed up access to it.

Benefits of Using XMLIndex Index

- XMLIndex index is:
 - Effective in any part of a query: SELECT list, the FROM list, and the WHERE clause of a query
 - Useful for XML fragment extraction
- Use an XMLIndex index:
 - With either XML schema-based or non-XML-schema-based data
 - For searches with XPath expressions that target collections
- XMLIndex index:
 - Indexes all possible XPath locations in your XML data
 - Supports path subsetting
 - Has synchronous and asynchronous modes of operation



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use the XMLIndex index for XPath expressions in the SELECT list, the FROM list, and the WHERE clause of a query. The XMLIndex index is useful for SQL functions, such as XMLQuery, XMLTable, XMLExists, XMLCast, extract, extractValue, and existsNode. XMLIndex index provides the following advantages over other indexing methods:

- XMLIndex is effective in any part of a query and is not limited to use in a WHERE clause.
- You do not need prior knowledge of the XPath expressions that are used in queries. XMLIndex is completely general. By default, XMLIndex indexes all possible XPath locations in your XML data.
- You can use XMLIndex for searches with XPath expressions that target collections, that is, nodes that occur multiple times within a document.
- The creation and maintenance of XMLIndex can be carried out in parallel by using multiple database processes.
- You can use an XMLIndex index with either XML schema-based or non-schema-based data. It can be used with unstructured storage, hybrid storage, and binary XML storage.

- For hybrid storage of XML schema-based data, `XMLIndex` can handle XPath expressions that target document fragments stored within a `CLOB` instance.
- You can update the `XMLIndex` indexes on binary XML storage in a piecewise manner, thereby considerably improving data manipulation language (DML) performance. This is not the case for other kinds of indexes that you might use with XML data.

The following are some of the other important features of `XMLIndex`:

- The `XMLIndex` index type supports path subsetting. With path subsetting, you can use XPath expressions to limit the nodes in an XML document that should be indexed. You can specify path subsetting either on an inclusive or on an exclusive basis. You learn more about path subsetting later in this lesson.
For example, you need this in document-oriented scenarios where tags related to formatting are omitted from the index.
- `XMLIndex` has synchronous and asynchronous modes of operation. In synchronous mode, an `XMLIndex` index operates just like any other database index. This means that the index maintenance takes place as part of the transaction that modifies the data. In asynchronous mode, the index maintenance takes place separately from the transaction that modifies the data. Applications thus avoid being impacted by the overhead associated with full XML indexing.
- You can use `XMLIndex` to discover information about the structure of the documents that have been indexed. This can be extremely helpful when attempting to perform an analysis of large amounts of unstructured XML content.

Structured and Unstructured Components of XMLIndex

An XMLIndex index can have the following two components:

- A structured component:
 - Used to index islands of predictable, structured data
- An unstructured component:
 - Used to index data that has little or variable structure



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLIndex is generally used to index semi-structured XML data (data that generally has little or no fixed structure). It applies to data that is stored using binary XML storage. However, semi-structured XML data can sometimes contain islands of predictable, structured data. Starting Oracle Database 11g Release 2, XMLIndex has capabilities to handle queries over structured components of XML as well.

Unlike a structured component, an unstructured component is general and relatively untargeted. Using an unstructured component alone can also lead to inefficiencies involving multiple probes and self-joins of its path table, for queries that project structured islands.

On the other hand, a structured component is not suited for queries that involve little structure or queries that extract XML fragments. Use a structured component to index structured islands of data; use an unstructured component to index data that has little structure.

XML Use Cases and XML Indexing



	Data-Centric	Document-Centric	
Use Case	XML schema-based data, with little variation and little structural change over time	Variable, free-form data, with some fixed embedded structures	Variable, free-form data
Typical Data	Employee record	Technical article with author, date, and title fields	Web document or book chapter
Storage Model	Object-Relational (Structured)	Binary XML	
Indexing	B-tree index	<ul style="list-style-type: none"> • XMLIndex with structured and unstructured components • XML Full-Text Index 	<ul style="list-style-type: none"> • XMLIndex with unstructured component index • XML Full-Text Index



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This diagram in the slide is the same diagram discussed in the lesson titled “Storing XML Data in Oracle XML DB.” In this lesson, we will focus on the indexing part. The last row indicates the applicability of XMLIndex for different XML data use cases. It shows that XMLIndex is appropriate for semi-structured XML data, and an XMLIndex index with a structured component is useful for document-centric data that contains structured islands.

Lesson Agenda

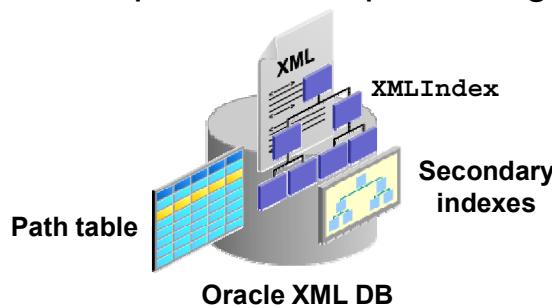
- Indexing XMLType data
- Using XMLIndex index
- **XMLIndex unstructured component**
- XMLIndex structured component
- Guidelines



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Logical Parts of the Unstructured Component of an XMLIndex

- The unstructured component of an XMLIndex is a logical index that has the following logical parts:
 - A path index
 - A value index
 - An order index
- Oracle XML DB implements XMLIndex by using a path table and a set of secondary indexes corresponding to its components to implement the preceding logical parts.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Logically, XMLIndex consists of:

- **The path index:** Indexes the XML tags of a document and identifies the document's various fragments based on simple (navigational) path expressions
- **The order index:** indexes the hierarchical positions of the nodes in an XML document. It keeps track of parent-child, ancestor-descendant, and sibling relations.
- **The value index:** This indexes the values of an XML document. It provides lookup by either value equality or value range. A value index is used for values in query predicates (WHERE clause).

The preceding three aspects of XMLIndex capture all relevant information about the original XML document. When you submit a query involving XPath, the XPath is decomposed into a SQL query that accesses the XMLIndex table. The generated query typically performs a set of path, value, and order-constrained lookups, and merges their results appropriately.

Logical Parts of the Unstructured Component of an XMLIndex

- Two secondary indexes are created automatically:
 - A *pikey index*, which implements the logical indexes for both *path* and *order*
 - A *real value index*, which implements the logical value index
- The path table contains one row for each indexed node in the XML document. For each indexed node, it stores:
 - The corresponding *rowid* of the table storing the document
 - A *locator*, which provides fast access to the corresponding document fragment
 - An *order key*, to record the hierarchical position of the node in the document
 - An *identifier* that represents an XPath path to the node
 - The effective *text value* of the node



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The unstructured component of an XMLIndex index uses a path table and a set of (local) secondary indexes on the path table, which implement the logical parts described on the previous page. Two secondary indexes are created automatically:

- A pikey index, which implements the logical indexes for both path and order. The pikey index handles paths and order relationships together, which gives the best performance in most cases.
- A real value index, which implements the logical value index

The base table on which the XMLIndex index is created owns the path table and the secondary indexes. The path table contains one row for each indexed node in the XML document. For each indexed node, the path table stores the following:

- The corresponding *rowid* of the table that stores the document
- A locator, which provides fast access to the corresponding document fragment. For binary XML storage of XML schema-based data, it also stores data-type information.
- An order key, to record the hierarchical position of the node in the document. You can think of this as a Dewey decimal key, such as the ones used in library cataloging and Internet protocol SNMP. In such a system, the key 3.21.5 represents the node position of the fifth child of the twenty-first child of the third child of the document root node.

- An identifier that represents an XPath path to the node
- The effective text value of the node

Note: Although you might create secondary indexes on path table columns, you can generally ignore the path table itself. You cannot access the path table, other than using a DESCRIBE statement and creating secondary indexes on it. You do not need to explicitly gather statistics on the path table. You need to only collect statistics on the XMLIndex index or the base table on which the XMLIndex index is defined.

For more information about the components of an XMLIndex index, see *Oracle XML DB Developer's Guide*.

Using XMLIndex with an Unstructured Component

- To include an unstructured component in an XMLIndex index, you use a path_table_clause in the PARAMETERS clause when you create or modify the XMLIndex index
- If you do not specify a structured component, then the index will have an unstructured component, even if you do not specify the path table.
- It is however generally a good idea to specify the path table, so that it has a recognizable, user-oriented name that you can refer to in other XMLIndex operations.
- If you do not name the path table then its name is generated by the system, using the index name you provide to CREATE INDEX as a base.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Creating an XMLIndex Index Unstructured Component

You can create an XMLIndex index by declaring the index type to be XDB.XMLIndex:

```
CREATE TABLE po_tab OF XMLType
XMLSCHEMA "http://localhost:8080/source/schemas/poSource/xsd/purchaseOrder.xsd"
ELEMENT "PurchaseOrder"
/
CREATE INDEX po_xmli ON po_tab(OBJECT_VALUE)
INDEXTYPE IS XDB.XMLINDEX
Parameters ('PATH TABLE po_path_table');
```

1

```
CREATE TABLE csx_tab OF XMLTYPE
XMLTYPE STORE AS BINARY XML
/
CREATE INDEX csx_xml_idx ON csx_tab(OBJECT_VALUE)
INDEXTYPE IS XDB.XMLINDEX
Parameters ('PATH TABLE csx_path_table');
```

2

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Example 1 shows an XMLIndex index created on a schema-based XMLType stored in the binary XML format (that is the default storage type, so you can omit that as in the second example).

```
Connected
CREATE TABLE succeeded.
CREATE INDEX succeeded.
```

Example 2 shows an XMLIndex index created on an XMLType table stored in the binary XML format.

```
Connected
CREATE TABLE succeeded.
CREATE INDEX succeeded.
```

Note: By default, the storage options of the XMLIndex path table and its secondary indexes are derived from the storage properties of the base table on which the XMLIndex is created. You can specify different storage options by using the PARAMETERS clause when creating an XMLIndex index.

Creating Secondary Indexes on the XMLIndex Unstructured Component

- In the CREATE INDEX statement, if you do not specify a secondary index for the VALUE column of the PATH TABLE, Oracle XML DB creates a default secondary index.
- You can create any number of additional secondary indexes on the VALUE column of the PATH TABLE.

Creating a function-based index on the path table is supported; whereas creating a function-based index on an XMLType column/table directly is deprecated.

```
CREATE INDEX fn_based_ix  
ON po_path_table(substr(VALUE, 1, 100));
```

Index created.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Creating Secondary Indexes

When you create an XMLIndex index, if you do not specify a secondary index for the VALUE column, Oracle XML DB creates a default secondary index on the VALUE column. This default index has the default properties. The default index is only for text data.

Like the name of the path table, the names of the secondary indexes on the path-table columns are generated automatically using the index name as a base, unless you specify them in the PARAMETERS clause.

You can create any number of additional secondary indexes on the VALUE column of the path table of an XMLIndex index. These secondary indexes can be function-based indexes and Oracle Text indexes. Depending on the appropriate type of index for a value, and whether it is cost-effective to use the index, when processing a query, Oracle database determines whether or not to use a given index for a given element occurrence.

The code example in the slide shows how to create a function-based index on the VALUE column of the path table by using the substr SQL function. This index can be useful if your queries often use the substr SQL function applied to the text nodes of XML elements.

Specifying Storage Options When Creating an XMLIndex Index

```
CREATE INDEX po_xmlindex_ix ON po_binxml (OBJECT_VALUE)
INDEXTYPE IS XDB.XMLIndex
PARAMETERS
('PATH TABLE po_path_table
(PCTFREE 5 PCTUSED 90 INITTRANS 5
STORAGE
(INITIAL 1k NEXT 2k MINEXTENTS 3 BUFFER_POOL KEEP)
NOLOGGING ENABLE ROW MOVEMENT PARALLEL 3)
PIKEY INDEX po_pikey_ix (LOGGING PCTFREE 1 INITTRANS 3)
VALUE INDEX po_value_ix (LOGGING PCTFREE 1 INITTRANS
3)');
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

By default, the storage options of a path table and its secondary indexes are derived from the storage properties of the base table on which the XMLIndex index is created. You can specify different storage options by using a PARAMETERS clause when you create the index, as shown in the code example in the slide. The PARAMETERS clause of CREATE INDEX (and ALTER INDEX) must be between single quotation marks (').

Note that, in addition to specifying storage options for the path table, the slide example names the secondary indexes on the path table.

Creating an XMLIndex Index Unstructured Component

If you do not name the path table then its name is generated by the system, using the index name you provide to CREATE INDEX as a base.

```
SELECT PATH_TABLE_NAME
FROM USER_XML_INDEXES
WHERE TABLE_NAME = 'PO_BINXML' AND INDEX_NAME =
  'PO_XMLINDEX_IX';
```

PATH_TABLE_NAME

SYS93433_PO_XMLINDE_PATH_TABLE



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Note

The slide example assumes that a table named `po_binxml` and an index named `po_binxml` are already created. The system generated name returned might be different.

Dictionary Views for XMLIndex

Query the following static public views for information about XMLIndex indexes:

- USER/ALL/DBA_XML_INDEXES

```
-- Run the notes page code example first to
-- create the po_xmli index.
```

```
SELECT PATH_TABLE_NAME
FROM USER_XML_INDEXES
WHERE TABLE_NAME = 'PO_TAB' AND INDEX_NAME =
'PO_XMLI';
```

PATH_TABLE_NAME
SYS93107_PO_XMLI_PATH_TABLE

SQL> ■



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When you use the DESCRIBE statement on any of the dictionary views mentioned in the slide, it provides information, such as the name of the base table on which the XMLIndex index is defined, the owner of the table, and the name of the index path table. If you do not name the path table during index creation, its name is generated by the system by using the index name that you provide to CREATE INDEX as a base. For example, you create an XMLIndex index by executing the following statement:

```
CREATE INDEX po_xmli ON po_tab(OBJECT_VALUE) INDEXTYPE IS
XDB.XMLIndex;
```

In this case, you can query the USER_XML_INDEXES view to find out the system-generated name of the path table as shown in example 1.

Determining the Names of the Secondary Indexes of an XMLIndex

```
SELECT INDEX_NAME, COLUMN_NAME, COLUMN_POSITION  
FROM USER_IND_COLUMNS  
WHERE TABLE_NAME IN  
  (SELECT PATH_TABLE_NAME  
   FROM USER_XML_INDEXES  
   WHERE INDEX_NAME = 'PO_XMLINDEX_IX')  
ORDER BY INDEX_NAME, COLUMN_NAME;
```

Subquery to find
the path table
name

INDEX_NAME	COLUMN_NAME	COLUMN_POSITION
SYS67563_PO_XMLINDE_PIKEY_IX	ORDER_KEY	3
SYS67563_PO_XMLINDE_PIKEY_IX	PATHID	2
SYS67563_PO_XMLINDE_PIKEY_IX	RID	1
SYS67563_PO_XMLINDE_VALUE_IX	SYS_NC00006\$	1

4 rows selected.

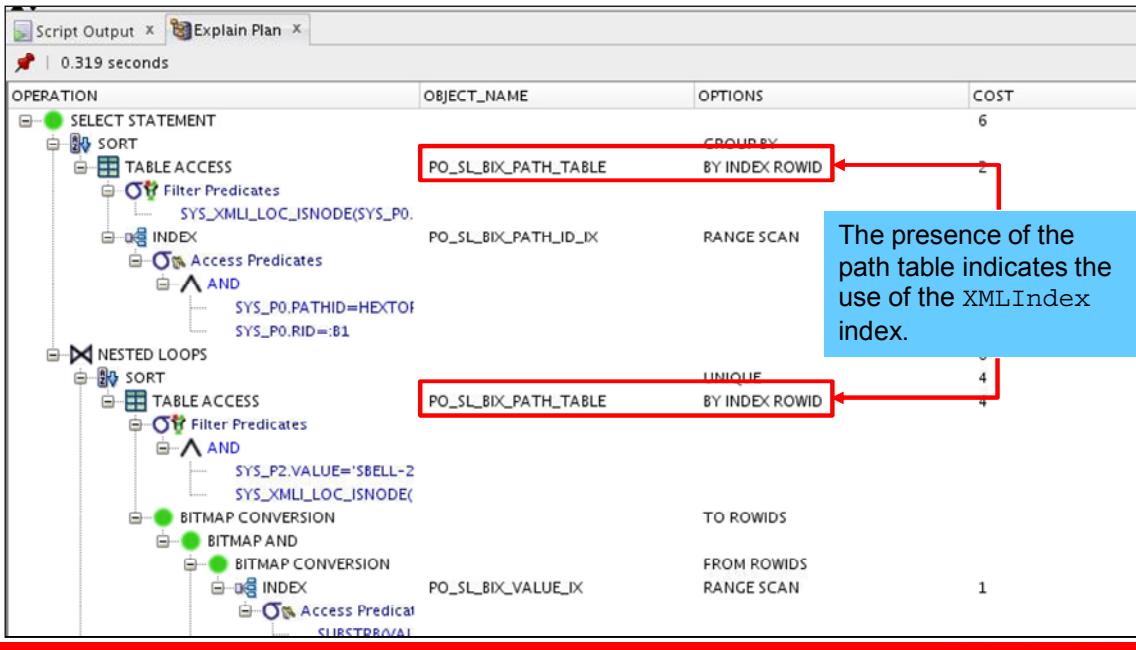
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Like the name of the path table, the names of the secondary indexes on the path-table columns are generated automatically using the index name as a base, unless you specify them in the PARAMETERS clause. The slide example shows how you can determine the names of system-generated secondary indexes using public view `USER_IND_COLUMNS`. It also shows that the pikey index uses three columns.

Determining XMLIndex Index Usage

Examine the Explain Plan to determine whether a particular XMLIndex index is used in resolving a query.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example Explain Plan shows the XMLIndex index usage.

If the XMLIndex index is used, its path table, order key, or path ID are referenced in the Explain Plan. The Explain Plan does not directly indicate that a domain index was used. It does not refer to the XMLIndex index by name.

The slide shows an example of using the XMLIndex index on an XQuery expression, and then examining the Explain Plan. The example shows a display of the Explain Plan by using the AUTOTRACE option in SQL Developer. For this example, you create an XMLType table for binary XML storage and insert the sample data as follows:

```
CREATE TABLE PO_SL_BIX_TABLE OF XMLType
XMLType STORE AS SECUREFILE BINARY XML;
INSERT INTO po_sl_bix_table
SELECT object_value
FROM purchaseorder;
```

```
CREATE TABLE succeeded.
132 rows inserted
```

First, you create an XMLIndex index, and then a secondary text index on the VALUE column of the path table.

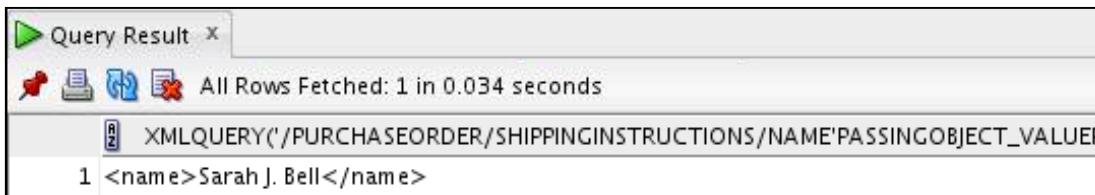
```
--Execute as oe/oe
CREATE INDEX po_sl_xmlindex_bix_ix ON po_sl_bix_table(object_value)
indextype is xdb.xmlindex
parameters ('PATH TABLE po_sl_bix_path_table PATH ID INDEX
po_sl_bix_path_id_ix
ORDER KEY INDEX po_sl_bix_order_key_ix
VALUE INDEX po_sl_bix_value_ix') ;
/
CREATE INDEX po_sl_bix_text_ix ON po_sl_bix_path_table
(VALUE) INDEXTYPE IS CTXSYS.CONTEXT
parameters ('transactional');
```

```
CREATE INDEX succeeded.
CREATE INDEX succeeded.
```

The Explain Plan in the slide shows the presence of the path table that indicates the use of the XMLIndex index. To see the execution plan, enter the following code in the SQL Worksheet, click the statement, and then press F10. The result is shown in the previous slide.

Example query:

```
SELECT XMLQuery('/PurchaseOrder/ShippingInstructions/name'
passing object_value returning content).getStringVal()
FROM PO_SL_BIX_TABLE WHERE
XMLEXISTS('/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
passing object_value);
```



XMLIndex Path Subsetting

- By default, XMLIndex indexes all the possible XPath locations in your XML data.
- You can narrow the focus of XMLIndex indexing by specifying a subset of all possible paths.
- XMLIndex path subsetting improves:
 - Index maintenance during DML operations
 - DDL performance
 - Query performance



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When you create an XMLIndex, by default, all the possible XPath locations in your XML data are indexed. However, you can explicitly specify the set of nodes that are to be indexed, thereby omitting the remaining nodes from the path table. This means that you can narrow the focus of XMLIndex indexing and improve performance.

Benefits of Using XMLIndex Path Subsetting

When you reduce the number of indexed nodes, the space usage of the XMLIndex is improved. Having fewer indexed nodes not only improves the data definition language (DDL) performance, but also improves the management efficiency of XMLIndex. A smaller number of indexed nodes improves data manipulation language (DML) performance. A smaller size for the path table makes queries that use XMLIndex more efficient.

Specifying Paths for XMLIndex

- Specify path subsetting when you create an XMLIndex index or modify an XMLIndex index.
- Use either of the following methods for path subsetting:
 - Path exclusion
 - Path inclusion
- Follow the path subsetting rules:
 - You cannot specify both path exclusion and path inclusion.
 - Index modification must either further restrict or further extend the path subset.
 - The paths must reference only child and descendent axes, and they must test only element and attribute nodes or their names.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can specify path subsetting either when you create an XMLIndex index by using CREATE INDEX, or when you modify it using ALTER INDEX. You specify the subsetting information in the PATHS parameter of the respective statement's PARAMETERS clause.

You can use either of the following methods for path subsetting:

- **Inclusion:** Start with an empty set of XPath expressions to be used in indexing and add paths to this inclusion set.
- **Exclusion:** Start with the default behavior of including all XPath expressions and exclude some of them from indexing.

You can use the INCLUDE keyword for inclusion and the EXCLUDE keyword for exclusion. For ALTER INDEX, the INCLUDE or EXCLUDE keyword is followed by the ADD or REMOVE keyword, to indicate whether the list of paths that follows the keyword is to be added or removed from the inclusion or exclusion list. You can also specify namespace mappings for the nodes targeted by the PATHS parameter.

If you create an index by using path exclusion, you can modify it by using only path exclusion. Similarly, if you create an index by using path inclusion, you can modify it by using only path inclusion.

Specifying Paths for XMLIndex: Examples

- Creating an XMLIndex index:

```
DROP INDEX DROP po_xmli;
CREATE INDEX po_tab_xmlindex_v2 ON po_tab (OBJECT_VALUE)
INDEXTYPE IS XDB.XMLINDEX
PARAMETERS ('PATHS (INCLUDE (/PurchaseOrder/LineItems//*
/PurchaseOrder/Reference))');
```

1

```
index PO_TAB_XMLINDEX_V2 created.
```

- Modifying an XMLIndex index:

```
ALTER INDEX po_tab_xmlindex_v2 REBUILD
PARAMETERS ('PATHS (INCLUDE ADD (/PurchaseOrder/Requestor
/PurchaseOrder/Actions/Action//*))');
```

2

```
index PO_TAB_XMLINDEX_V2 altered.
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In Example 1, a `po_tab_xmlindex_v2` index is created that indexes only the top-level `PurchaseOrder` element and some of its children, as follows:

- All `LineItems` elements and their descendants
- All `Reference` elements

The statement starts with an empty set of XPath expressions to be used in indexing and adds the specified paths to the inclusion set.

Example 2 adds two more paths to those used for indexing. These paths index the `Requestor` element and the descendants of the `Action` element.

If an XPath expression that is to be used for XMLIndex indexing uses namespace prefixes, you can use a `NAMESPACE MAPPING` clause to the `PATHS` list to specify those prefixes.

XMLIndex Parallelism

- Use the PARALLEL clause when you create or alter an XMLIndex index to ensure that the database server creates and maintains the index in parallel.
- XMLIndex parallelism improves the performance of index creation and maintenance.
- Example of creating an XMLIndex index in parallel:

```
CREATE INDEX po_xmlindex_ix_v3 ON sale_info  
(sale_po_clob) INDEXTYPE IS XDB.XMLIndex  
PARALLEL 10;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When you use the PARALLEL clause (with optional degree) while creating or altering an XMLIndex index, the database server creates and maintains the index in parallel. Because multiple processes work together to create the index, the database server creates the index more quickly than if a single server process created the index sequentially. XMLIndex parallelism also consumes more storage because each query process uses storage parameters separately.

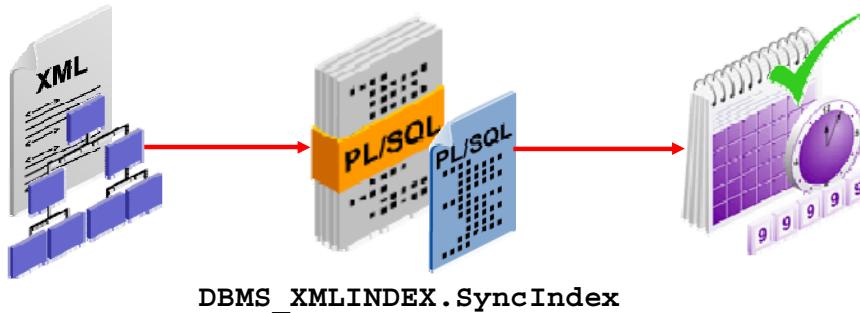
The slide example shows that the database server creates the path table and the secondary indexes with the same parallelism degree as the XMLIndex index itself (that is, a degree of 10 by inheritance). You can specify different parallelism degrees for these by using separate PARALLEL clauses.

Asynchronous Maintenance of XMLIndex Index

- You can specify deferment of index maintenance by using the PARAMETERS clause of a CREATE INDEX or ALTER INDEX statement.

```
CREATE INDEX po_demo_idx ON po_demo (OBJECT_VALUE)
INDEXTYPE IS XDB.XMLIndex
PARAMETERS ('ASYNC (SYNC EVERY "FREQ=HOURLY; INTERVAL = 1")');
```

- You can manually synchronize the index by using the DBMS_XMLINDEX.SYNCINDEX PL/SQL procedure.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

By default, XMLIndex indexing is maintained at each DML operation, so that it remains synchronized with the base table. In some situations, the cost of immediate maintenance is not acceptable and using possibly stale indexes is adequate. You may prefer to defer the cost of index maintenance to a convenient time, for example, upon commit of each transaction or at some time when the database load is reduced. This is known as asynchronous maintenance of the XMLIndex index. Asynchronous maintenance of XMLIndex indexes can provide the following benefits:

- Improved DML performance
- Improved performance of index maintenance by enabling bulk-loading of unsynchronized rows when an index is synchronized

To specify deferment index maintenance, you can use the ASYNC clause in the PARAMETERS clause of a CREATE INDEX or ALTER INDEX statement for XMLIndex.

The slide example specifies deferred synchronization for an XMLIndex index.

Note: To use EVERY, you must have the CREATE JOB privilege.

Use the following example to manually synchronize the index created in the slide example:

```
EXEC DBMS_XMLINDEX.SyncIndex ('OE', 'PO_DEMO_IDX');
```

Lesson Agenda

- Indexing XMLType data
- Using XMLIndex index
- XMLIndex unstructured component
- XMLIndex structured component
 - Creating an XMLIndex with a structured component
 - Using a structured XMLIndex component for query
- Guidelines



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLIndex Structured Component

XMLIndex structured component:

- Is used for queries that project fixed, structured islands of XML content
- Is an alternative to using multiple function-based indexes
- Internally organizes the structured islands of data in a relational format
- Is similar to the XMLTable SQL function



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLIndex structured component is appropriate for queries that project fixed, structured islands of XML content, even if the surrounding data is relatively unstructured. It is not suited for queries that involve little structure or queries that extract XML fragments.

Internally, the structured component of an XMLIndex organizes data in a relational format. In this, XMLIndex structured component is similar to XMLTable SQL function.

The structured component of an XMLIndex index can also be viewed as a generalized function-based index. A function-based index is similar to a structured XMLIndex component that has only one relational column. Instead of creating multiple function-based indexes, you can consider a structured XMLIndex index along with secondary B-tree indexes on the columns of the structured XMLIndex index content table.

Creating an XMLIndex Index with Structured Component

```

CREATE INDEX po_xmlindex_ix ON po_clob (OBJECT_VALUE)
INDEXTYPE IS XDB.XMLIndex PARAMETERS
('PATH TABLE path_tab');

BEGIN
DBMS_XMLINDEX.registerParameter
    ('myparam','ADD_GROUP GROUP po_item
XMLTable po_idx_tab ''/PurchaseOrder''
COLUMNS reference VARCHAR2(30) PATH ''Reference'',...
lineitem XMLType PATH ''LineItems/LineItem'' VIRTUAL
XMLTable po_index_lineitem ''/LineItem'' PASSING lineitem
COLUMNS itemno BINARY_DOUBLE PATH ''@ItemNumber'',
...
unitprice BINARY_DOUBLE PATH ''Part/@UnitPrice''');
END;
/
ALTER INDEX po_xmlindex_ix PARAMETERS ('PARAM myparam');

```

```

CREATE TABLE succeeded.
CREATE INDEX succeeded.
anonymous block completed

```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To include a structured component in an XMLIndex index, you use a `structured_clause` in the `PARAMETERS` clause when you create or modify the XMLIndex index.

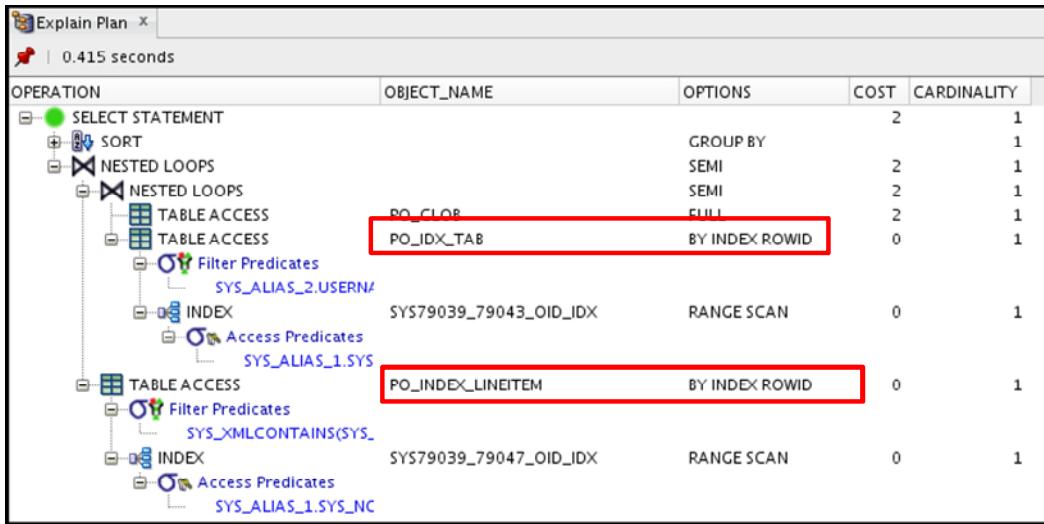
In an `ALTER INDEX` statement, when you precede the `GROUP` keyword with the modification operation keyword `ADD_GROUP`, it specifies a new group (island) to be added to the index. Similarly, you can use the `DROP_GROUP` modification operation keyword with the `GROUP` keyword to delete a group.

Example 1 in the slide shows the creation of an XMLIndex index with only an unstructured component (`PARAMETERS` clause explicitly names the path table).

In Example 2, you use `DBMS_XMLINDEX.registerParameter` to register `myparam`. `ADD_GROUP` specifies a new structured island. This structured group includes two relational tables `po_idx_tab` (top-level table) and `po_index_lineitem` (second-level table), each specified with `XMLTable` keyword. The `VIRTUAL` keyword specifies that the `XMLType` column `lineitem` itself is not materialized. The column data is stored in the XMLIndex index only in the form of relational columns specified by its corresponding `XMLTable` table. You cannot create more than one `XMLType` column in a given `XMLTable` clause. Therefore, you must define an additional group. This example uses the `ALTER INDEX` statement to update the index parameters to include those in the `myparam` parameter string.

Using A Structured XMLIndex Component for Query

```
SELECT XMLQuery('/PurchaseOrder/LineItems/LineItem' PASSING
OBJECT_VALUE RETURNING CONTENT) FROM po_clob
WHERE XMLExists('/PurchaseOrder/LineItems/LineItem
[ora:contains (Description, "Picnic") > 0]' PASSING OBJECT_VALUE)
AND XMLExists('/PurchaseOrder [User="SBELL"]' PASSING
OBJECT_VALUE);
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If the structured component of the index is used, then one or more of its index content tables is called out in the execution plan. The example in the slide shows an execution plan that indicates that the XMLIndex index created in the previous slide is picked up for a query that uses two WHERE clause predicates. With only the unstructured XMLIndex component, this query would involve a join of the path table to itself, because of the two different paths in the WHERE clause.

Note: To display the execution plan, place your cursor on the query, select the OE schema, and then press F10.

Using XMLIndex: Guidelines

The following are some guidelines for using XMLIndex:

- Avoid prefixing // with ancestors elements.
- Avoid prefixing /* with ancestors elements.
- Use count (*) in a SELECT clause whenever possible.
- Reduce the number of XPath expressions used in a query FROM list as much as possible.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The following examples are applicable only when the two alternatives return the same result set.

Examples:

- Use //c, *not /a/b//c*, provided that these return the same result set.
- Use /*/*/*, *not /a/*/**, provided that these return the same result set.
- If you know that a LineItem element in a purchase order document has only one Description child, use the following query:

```
SELECT count(*) FROM po_clob, XMLTable('//LineItem' PASSING  
OBJECT_VALUE);
```

But do not use the following query:

```
SELECT count(li.value)  
FROM po_clob p, XMLTable('//LineItem' PASSING p.OBJECT_VALUE  
COLUMNS value VARCHAR2(30) PATH  
'/LineItem/Description') li;
```

Note that this is not a complete list of guidelines for using XMLIndex.

Turning Off XMLIndex

You can use either of the following optimizer hints to turn off the use of XMLIndex:

- /*+ NO_XMLINDEX_REWRITE */
- /*+ NO_XMLINDEX_REWRITE_IN_SELECT */
- /*+ NO_XML_QUERY_REWRITE */

Examples:

```
SELECT /*+ NO_XMLINDEX_REWRITE */
count(*) FROM po_clob WHERE existsNode(OBJECT_VALUE,
'/*') = 1;

SELECT /*+ NO_XML_QUERY_REWRITE */
count(*) FROM po_clob WHERE existsNode(OBJECT_VALUE,
'/*') = 1;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Each of the optimizer hints in the slide turns off the use of all XMLIndex indexes. In addition to turning off the use of XMLIndex index, NO_XML_QUERY_REWRITE turns off all XPath rewrites.

The NO_XMLINDEX_REWRITE_IN_SELECT optimizer hint turns off the use of XMLIndex indexes only for XPath expressions in the SELECT list. This means that XMLIndex indexes can still be used for XPath expressions in other query parts, such as a WHERE clause or a FROM clause. You will find this optimizer hint especially useful with XML data that is stored as binary XML, in cases where streaming evaluation of the XPath expressions in a SELECT list provides better performance than XMLIndex.

Note: Oracle XML DB rewrites SQL statements that contain XPath expressions to purely relational SQL statements, which can be processed efficiently. This is known as XPath rewrite.

Summary

In this lesson, you should have learned how to:

- Create a function-based index on XMLType data
- Create an XMLIndex index
- Identify the components of XMLIndex
- View XMLIndex information in a data dictionary
- Determine XMLIndex index usage
- Specify the paths for XMLIndex index
- Use optimizer hints to turn off XMLIndex



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 11: Overview

This practice covers the following topics:

- Creating an XMLIndex index and determining its usage
- Querying data dictionary views for XMLIndex



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

12

Generating XML Data

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Use XQuery to generate XML
- Use standard SQL/XML functions to generate XML
- Use Oracle SQL functions to generate XML
- Use the `DBMS_XMLGEN` PL/SQL package to generate XML



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

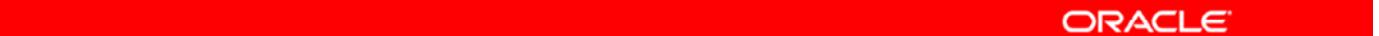
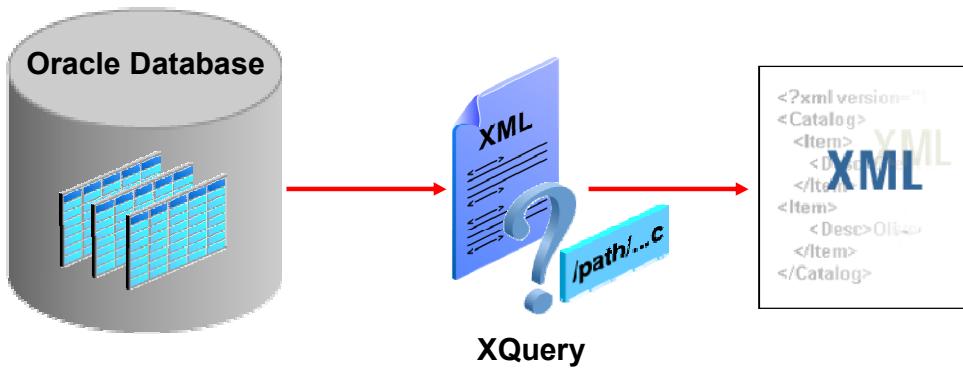
Lesson Agenda

- Generating XML data by:
 - Using XQuery
 - Using the SQL/XML standard functions
 - Using the Oracle Database SQL functions
 - XMLCOLATTVAL
 - SYS_XMLAGG
 - XMLCDATA
 - XMLROOT
 - Using the DBMS_XMLGEN PL/SQL package



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Using XQuery to Generate (Construct) XML Data from the Relational Data



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use XQuery to construct XML data from relational data. You can use `XMLQuery` SQL function to query existing relational tables or views and construct XML data at run time, without explicitly creating XML views over the relational data.

However, the two approaches to generating XML (using SQL/XML functions and using XQuery) are not substitutes for each other. You use SQL/XML functions for a SQL-centric query and XQuery for an XML-centric query.

List of the SQL/XML Functions

- Use the following SQL/XML functions to generate XML data from the result of a SQL query:
 - XMLEMENT
 - XMLATTRIBUTES
 - XMLFOREST
 - XMLCONCAT
 - XMLAGG
 - XMLSERIALIZE
 - XMLCOMMENT, XMLPI, and XMLPARSE
- Use the following SQL/XML functions to query and access XML content as part of normal SQL operations:
 - XMLQuery, XMLTable, XMLExists, and XMLCast



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide contains a list of the SQL/XML functions. The first list contains the SQL/XML functions that you can use to generate XML data from the result of a SQL query. These functions are covered in this lesson. The second list contains the SQL/XML functions that you can use to query and access XML content as part of normal SQL operations. This is covered in the lesson titled “Using XQuery to Retrieve XML Data in Oracle XML DB.”

Note: All of the XML-generation SQL functions convert scalars and user-defined data-type instances to their canonical XML format. In this canonical mapping, user-defined data-type attributes are mapped to XML elements.

Generating an XML Document from Relational Data: Example

```
-- For the complete code, see the code_12_06_s.sql script.
-- Creates the /public/employees folder on the XML DB repository. It also
-- creates and populates the employees.xml and departments.xml resources.

DECLARE
XMLDOC XMLType;
BEGIN
SELECT XMLQuery( 'for $j in 1
return ( <EMPLOYEES>
    { for $i in fn:collection("oradb:/HR/EMPLOYEES") /ROW
      where $i/EMPLOYEE_ID <= 102
      return ( <EMPLOYEE>
<EMPNO>{xs:string($i/EMPLOYEE_ID)}</EMPNO>
<ENAME>{xs:string($i/LAST_NAME)}</ENAME>
<SAL>{xs:integer($i/SALARY)}</SAL>
</EMPLOYEE>)
    </EMPLOYEES>)'
RETURNING CONTENT) INTO XMLDOC
FROM DUAL;
...
END;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The partial XMLQuery code in the slide shows how `fn:collection` can be used to construct an XML document from the data stored in the `EMPLOYEES` relational table. The partial output is the following:

```
<EMPLOYEES>
  <EMPLOYEE>
    <EMPNO>100</EMPNO><ENAME>King</ENAME><SAL>24000</SAL>
  </EMPLOYEE>
  ...
  
```

The two input parameters, `hr` and `EMPLOYEES`, tell `fn:collection` to query the `EMPLOYEES` table in the `hr` database schema. `fn:collection` returns a sequence of XML documents representing the rows of the `HR.employees` table. By specifying `$i/EMPLOYEE_ID <= 102` in the `where` clause of the `FLWOR` expression, the result sequence is restricted to only the first three employees.

To cast the SQL values to the appropriate types for the XML elements, the XQuery functions `xs:string()` and `xs:integer()` are used in the `return` clause of the `FLWOR` expression.

Lesson Agenda

- Generating XML data by:
 - Using XQuery
 - Using the SQL/XML standard functions
 - Using the Oracle Database SQL functions
 - XMLCOLATTVAL
 - SYS_XMLAGG
 - XMLCDATA
 - XMLROOT
 - Using the DBMS_XMLGEN PL/SQL package



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLELEMENT Function

-- Syntax

```
XMLELEMENT ([NAME] identifier
            [, XML_attributes_clause]
            [, value_expr]...)
```

-- Example

```
SELECT XMLELEMENT("Employee_Name", first_name || ' ' || last_name)
  AS result
  FROM employees
 WHERE employee_id > 200;
```

New XML element name New XML element content

RESULT

```
<Employee_Name>Michael Hartstein</Employee_Name>
<Employee_Name>Pat Fay</Employee_Name>
<Employee_Name>Susan Mavris</Employee_Name>
<Employee_Name>Hermann Baer</Employee_Name>
<Employee_Name>Shelley Higgins</Employee_Name>
<Employee_Name>William Gietz</Employee_Name>
```

6 rows selected

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLELEMENT function is used to create an XML element from relational data. The XMLELEMENT function takes three arguments:

1. A name identifier that specifies the name of the XML element to be created. The identifier does not have to be a column name, but it cannot be an expression. Use double quotation marks around the identifier name to preserve its case.
2. An optional collection of attributes for the element, called an XML attributes clause
3. Zero or more arguments that make up the element's content. If a value expression evaluates to NULL, no content is created for that argument.

The XMLELEMENT function returns an instance of XMLType.

The slide shows an example of generating an Employee_Name element for those employees with employee_id greater than 200. The values of the first_name and last_name columns are concatenated and used as the content of the Employee_Name element.

XMLATTRIBUTES Function

-- Syntax

```
XMLATTRIBUTES (value_expr
[, value_expr]...)
```

-- Create attributes for the Employee_Name element for those
-- employees whose job_id is ST_CLERK.

```
SELECT XMLEMENT ("Employee_Name",
    XMLATTRIBUTES(employee_id AS id, 'ST_CLERK' AS job)
    ,first_name||' '||last_name)
AS result
FROM employees WHERE job_id= 'ST_CLERK';
```

RESULT

```
<Employee_Name ID="125" JOB="ST_CLERK">Julia Nayer</Employee_Name>
<Employee_Name ID="126" JOB="ST_CLERK">Irene Mikkilineni</Employee_Name>
<Employee_Name ID="125" JOB="ST_CLERK">Julia Nayer</Employee_Name>
```

...

```
<Employee_Name ID="143" JOB="ST_CLERK">Randall Matos</Employee_Name>
<Employee_Name ID="144" JOB="ST_CLERK">Peter Vargas</Employee_Name>
```

20 rows selected

→ Second attribute (2)

→ First attribute (1)

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLATTRIBUTES function is used along with the XMLEMENT function to create XML attributes. It is used in the second (optional) XML attributes clause argument of the XMLEMENT function. The XMLATTRIBUTES function accepts a comma-separated list of arguments. Each argument can be a column name, a column expression, or a literal value. An alias, optionally preceded by an AS keyword, must be specified for a column expression and a literal value.

The column name or alias specified is used as the name of the attribute. The attribute value is set to the column value, the result of the column expression, or the literal value. If an expression evaluates to the NULL value, the attribute is excluded from the element.

The example in the slide creates attributes for the Employee_Name element for those employees whose job_id is ST_CLERK.

1. The first attribute is specified as a column name with ID as its alias.
2. The second attribute is specified as a literal value with the alias JOB.

XMLFOREST Function

-- Syntax

```
XMLFOREST (value_expr
            [, value_expr]...)
```

-- Example

```
SELECT XMLFOREST(first_name, last_name)
AS result
FROM employees;
```

New XML element 2

New XML element name 1

The screenshot shows the Oracle SQL Script Output window. It has a toolbar at the top with icons for script, edit, run, and save. Below the toolbar, it says "Task completed in 0.581 seconds". The main area is titled "RESULT" and contains the following XML output:

```
<FIRST_NAME>E1len</FIRST_NAME><LAST_NAME>Abe1</LAST_NAME>
<FIRST_NAME>Sundar</FIRST_NAME><LAST_NAME>Ande</LAST_NAME>
...
<FIRST_NAME>Eleni</FIRST_NAME><LAST_NAME>Zlotkey</LAST_NAME>
```

At the bottom of the output area, it says "107 rows selected".

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLFOREST function produces an XML element for each value expression that is specified in its comma-separated list of arguments. The result is called a forest. The arguments of the XMLFOREST function can be a column name, a column expression, an object type instance, or a literal value. The XML element name for each argument is derived from the column name or an optional alias. The alias is required if an argument is a column expression, object type, or a literal value. The alias is optional if an argument is a simple column name.

If a value expression evaluates to NULL, an element is not created for that expression. The XMLFOREST function can be used as a stand-alone function, nested within itself, or nested within an XMLELEMENT function that provides the root element for the forest.

The example in the slide creates XML elements from the `first_name` and `last_name` columns, for all employees.

Generating Nested XML Elements

```
-- Using the XMLELEMENT function:

SELECT XMLELEMENT("Emp",
    XMLELEMENT("Employee_id", employee_id),
    XMLELEMENT("Salary", salary),
    XMLELEMENT("Hire_Date", hire_date))
AS result
FROM employees
WHERE employee_id > 200;
```

```
-- Using the XMLFOREST function:

SELECT XMLELEMENT("Emp",
    XMLFOREST(employee_id "Employee_id", salary "Salary",
        hire_date "Hire_Date"))
AS result
FROM employees
WHERE employee_id > 200;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLELEMENT function can be nested to produce XML data with a nested structure. However, the XMLFOREST function is better suited for the task.

The slide shows examples to generate the same output by using the XMLELEMENT and XMLFOREST functions. The output for both the queries is:

A screenshot of the Oracle SQL Developer interface, specifically the 'Script Output' window. The window title is 'Script Output'. It shows the following XML output:

```
<Emp><Employee_id>201</Employee_id><Salary>13000</Salary><Hire_Date>2004-02-17</Hire_Date></Emp>
<Emp><Employee_id>202</Employee_id><Salary>6000</Salary><Hire_Date>2005-08-17</Hire_Date></Emp>
<Emp><Employee_id>203</Employee_id><Salary>6500</Salary><Hire_Date>2002-06-07</Hire_Date></Emp>
<Emp><Employee_id>204</Employee_id><Salary>10000</Salary><Hire_Date>2002-06-07</Hire_Date></Emp>
<Emp><Employee_id>205</Employee_id><Salary>12008</Salary><Hire_Date>2002-06-07</Hire_Date></Emp>
<Emp><Employee_id>206</Employee_id><Salary>8300</Salary><Hire_Date>2002-06-07</Hire_Date></Emp>
```

Below the XML output, the message '6 rows selected' is displayed.

Using the XMLCONCAT Function

```
-- Syntax
```

```
XMLCONCAT (XMLType_instance  
[, XMLType_instance]...)
```

```
-- Example
```

```
SELECT XMLEMENT ("Full_Name",  
    XMLATTRIBUTES(employee_id),  
    XMLCONCAT(XMLELEMENT("First", first_name),  
        XMLType('<Middle>James</Middle>'),  
        XMLELEMENT("Last", last_name)))  
AS result  
FROM employees  
WHERE employee_id=200;
```

```
RESULT
```

```
<Full_Name EMPLOYEE_ID="200"><First>Jennifer</First><Middle>James</Middle>  
<Last>Whalen</Last></Full_Name>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLCONCAT function combines all the arguments passed to it and creates an XML fragment. It takes as arguments a series of XMLType instances and returns an XMLType value. The example in the slide creates a Full_Name element for the employee with ID 200. The XMLCONCAT function is used to concatenate three XMLType instances:

- A First element created by using the XMLEMENT function
- A Middle element
- A Last element created by using the XMLEMENT function

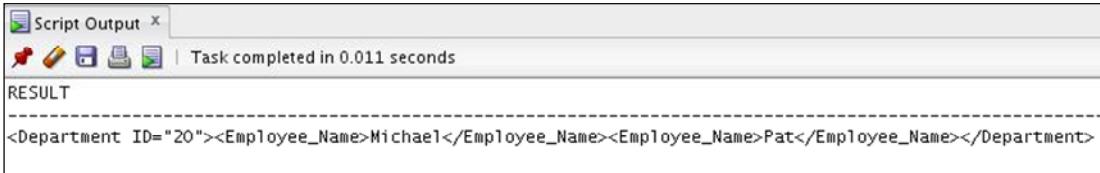
Using the XMLAGG Function

-- Syntax

```
XMLAGG (XMLType_instance  
[ORDER BY sort_list])
```

-- Example

```
SELECT      XMLEMENT ("Department",  
                      XMLATTRIBUTES ('20' "ID"),  
                      XMLAGG(XMLEMENT ("Employee_Name",  
                            first_name) ORDER BY first_name))  
AS result  
FROM employees  
WHERE department_id=20;
```



The screenshot shows the Oracle SQL Developer interface. A script has been run, and the output is displayed in the 'Script Output' window. The window title is 'Script Output X'. It shows a toolbar with icons for script, edit, run, and refresh. Below the toolbar, a message says 'Task completed in 0.011 seconds'. The main area is labeled 'RESULT' and contains the XML output of the query:

```
<Department ID="20"><Employee_Name>Michael</Employee_Name><Employee_Name>Pat</Employee_Name></Department>
```



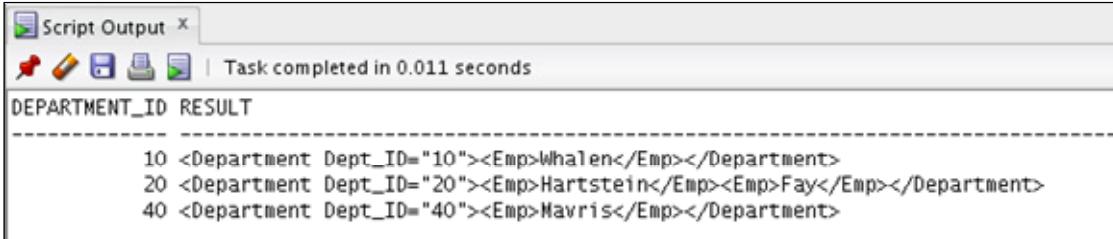
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLAGG function is used to combine XMLType instances across multiple rows. It is a SQL group function that produces a forest of XML elements from a set of XML elements. Any argument with a NULL value is excluded from the result. Within the ORDER BY clause, Oracle Database does not interpret number literals as column positions, as it does in other uses of this clause. It simply interprets them as number literals.

The slide example generates a root element Department with an attribute ID. The child elements for the Department element are created by using the XMLAGG function. The XMLAGG function aggregates the first names of the employees belonging to department 20 and returns a set of Employee_Name elements from multiple rows.

Generating Master-Detail Content

```
-- Generate XML elements for departments 10, 20, and 40 that  
-- contain the names of employees in the respective  
-- departments.  
  
SELECT    department_id,  
          XMLELEMENT("Department",  
          XMLATTRIBUTES(department_id "Dept_ID"),  
          XMLAGG(XMLELEMENT("Emp", last_name))) AS result  
FROM employees  
WHERE department_id in (10, 20, 40)  
GROUP BY department_id;
```



The screenshot shows the Oracle SQL Developer interface with a 'Script Output' window. The window title is 'Script Output X'. It displays the query results for 'DEPARTMENT_ID RESULT'. The output shows three XML documents corresponding to department IDs 10, 20, and 40. Each document contains one or more employee names under the 'Emp' element.

```
10 <Department Dept_ID="10"><Emp>Whalen</Emp></Department>  
20 <Department Dept_ID="20"><Emp>Hartstein</Emp><Emp>Fay</Emp></Department>  
40 <Department Dept_ID="40"><Emp>Mavris</Emp></Department>
```



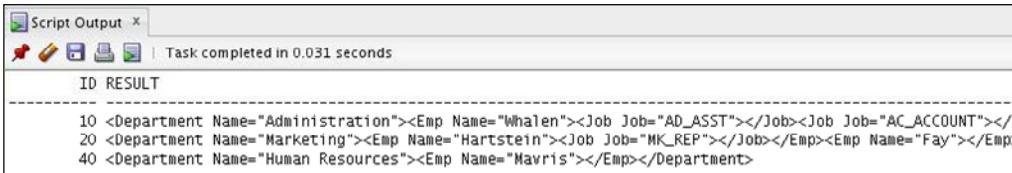
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide produces a `Department` element for each row returned by the query. Each `Department` element contains an `Emp` child element for each employee belonging to the department rows returned by the query.

Note: The example must use a `GROUP BY` clause because the `XMLAGG` function is a SQL group function that is used with the single-valued `department_id` column, which is used in the `SELECT` list and as an argument in the `XMLATTRIBUTES` function.

A Complex Master-Detail: Example

```
-- Generate XML elements of each employee with job history  
-- within the department.  
  
SELECT department_id id, XMLELEMENT("Department",  
    XMLATTRIBUTES(d.department_name "Name"),  
    (SELECT XMLAGG(XMLELEMENT("Emp",  
        XMLATTRIBUTES(e.last_name "Name"),  
        (SELECT XMLAGG(XMLELEMENT("Job",  
            XMLATTRIBUTES(j.job_id "Job"))  
            FROM job_history j  
            WHERE j.employee_id=e.employee_id)))  
        FROM employees e  
        WHERE e.department_id=d.department_id))  
AS result  
FROM departments d  
WHERE department_id in (10,20,40);
```



The screenshot shows the Oracle SQL Developer interface with the 'Script Output' tab selected. The output window displays the results of the executed SQL query. The results are grouped by department ID (10, 20, 40) and show the department name, employee names, and their job histories.

ID	RESULT
10	<Department Name="Administration"><Emp Name="Whalen"><Job Job="AD_ASST"></Job><Job Job="AC_ACCOUNT"></Job>
20	<Department Name="Marketing"><Emp Name="Hartstein"><Job Job="MK_REP"></Job><Emp Name="Fay"></Emp>
40	<Department Name="Human Resources"><Emp Name="Mavris"></Emp></Department>

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLAGG function can be used to reflect the hierarchical nature of some relationships that exist in tables. The example in the slide generates department elements that contain an element for each employee. Within each employee element is his or her job history.

XMLSERIALIZE Function

```
XMLSERIALIZE ({DOCUMENT|CONTENT}
               value_expr [AS datatype]
               [ENCODING xml_encoding]
               [VERSION string_literal]
               [NO INDENT|INDENT [SIZE=number]]
               [HIDE|SHOW DEFAULTS])
```

```
-- Use the XMLSERIALIZE function to pretty-print the output.
SELECT XMLSERIALIZE(CONTENT XMLEMENT ("Department",
    XMLATTRIBUTES('20' "ID"),
    XMLAGG(XMLELEMENT ("Employee_Name",
        first_name) order by first_name))
    INDENT SIZE=2) AS result
FROM employees
WHERE department_id=20;
```

The screenshot shows the Oracle SQL Developer interface. In the top left, there's a toolbar with icons for script, execute, and save. Next to it is a status bar that says 'Task completed in 0.006 seconds'. Below the toolbar is a section labeled 'RESULT' containing the output of the query. The output is a single line of XML:

```
<Department ID="20">
  <Employee_Name>Michael</Employee_Name>
  <Employee_Name>Pat</Employee_Name>
</Department>
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLSERIALIZE function creates a string or large object (LOB) that contains the contents of the argument value expression. In the syntax, note the following:

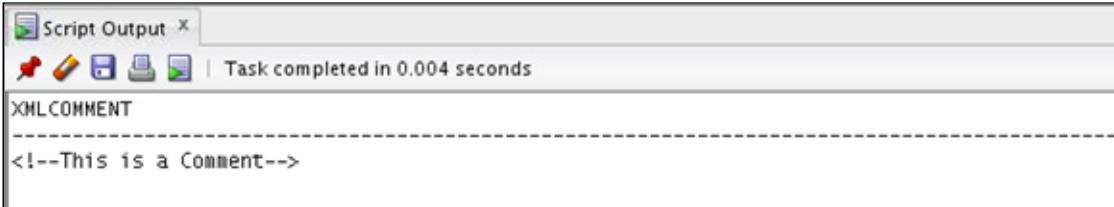
- If you specify DOCUMENT, the result of evaluating the value expression must be a well formed single-rooted XML document.
- If you specify CONTENT, the result of the value expression must correspond only to a well formed XML fragment.

If specified, datatype must be either VARCHAR, VARCHAR2, CLOB, or BLOB. If the datatype is not specified, the default is CLOB. The ENCODING clause is used if the datatype is specified as BLOB. The VERSION clause is used to specify the XML version in the XML declaration statement. The INDENT clause is used to pretty-print the output. SHOW DEFAULTS and HIDE DEFAULTS apply only to the XML schema-based data.

The XMLCOMMENT Function

```
-- XMLCOMMENT syntax. The value returned by the function  
-- takes the following form: <!--string-->  
  
XMLCOMMENT (value_expr)
```

```
-- The following code example shows how to generate an XML  
-- comment by using the XMLCOMMENT function.  
  
SELECT XMLCOMMENT('This is a Comment') AS "XMLCOMMENT"  
FROM DUAL;
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLCOMMENT function generates an XML comment by using an evaluated result of the value expression. This result is used as the body of the generated XML comment. The XMLCOMMENT function returns an instance of XMLType. If the evaluated result of the value expression is NULL, the function returns NULL. However, the function raises an error if the constructed XML is not a legal XML comment. In particular, the evaluated string result must not contain two consecutive hyphens. The value returned by the function takes the following form: <!--string-->.

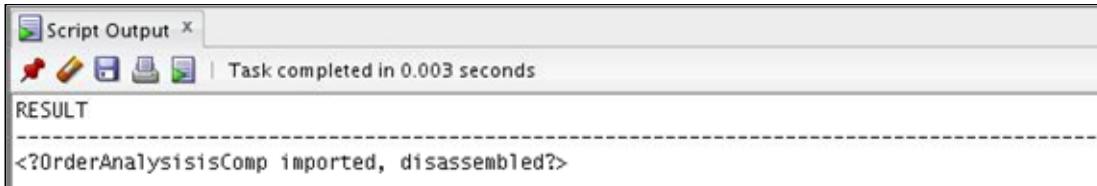
The XMLPI Function

```
-- XMLPI Syntax

XMLPI ([ NAME ] identifier
       [, value_expr] )
```

```
-- The following example shows how to use the XMLPI
-- function to generate the processing instruction.

SELECT XMLPI(NAME "OrderAnalysisisComp",'imported,
             disassembled') AS Result
FROM DUAL;
```



```
Script Output X
Task completed in 0.003 seconds
RESULT
<?OrderAnalysisisComp imported, disassembled?>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLPI function is used to create an XML-processing instruction. A processing instruction (PI) is commonly used to provide additional information to the application. The application uses the PI to determine how best to process the XML document. An example of a PI is the XML style sheet that can be used to inform an application that the current XML document should be transformed by using the specified Extensible Stylesheet Language (XSL) document. The XMLPI function generates XML processing instructions by using identifier and (optionally) the evaluated result of the value expression. The XMLPI function returns an instance of XMLType.

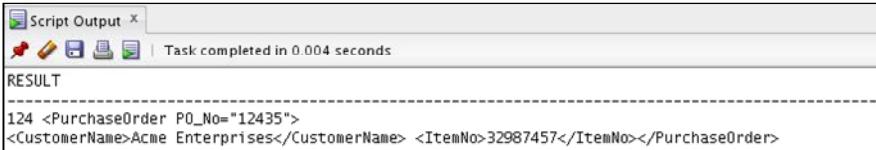
The XMLPI function raises an error if the constructed XML is not a legal XML processing instruction. For example, the function has the following restrictions:

- Identifier must not be the word `xml` (uppercase, lowercase, or mixed case).
- The evaluated string result must not contain the character sequence `?>`.

The XMLPARSE Function

```
-- XMLPARSE syntax:  
  
XMLPARSE ({ DOCUMENT | CONTENT }  
           value_expr [ WELLFORMED ] )
```

```
-- The following example uses the XMLPARSE function to parse  
-- the string containing XML data and returns an XMLType  
-- instance. In the query, CONTENT indicates that the input  
-- need not be single-rooted.  
  
SELECT XMLPARSE( CONTENT  
  '124 <PurchaseOrder PO_No="12435">  
  <CustomerName>Acme Enterprises</CustomerName>  
  <ItemNo>32987457</ItemNo></PurchaseOrder>' WELLFORMED)  
AS Result  
FROM DUAL;
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLPARSE function parses a string containing XML data and generates a corresponding value of XMLType. In the syntax, you use the keyword WELLFORMED to inform the parser that the argument value expression is well formed. This means that Oracle XML DB will not perform checks to ensure that the document is well formed.

- If you specify DOCUMENT, the value expression must correspond to a single-rooted well-formed XML document.
- If you specify CONTENT, the value expression must correspond only to a well-formed XML fragment.

Lesson Agenda

- Generating XML data by:
 - Using XQuery
 - Using the SQL/XML standard functions
 - Using the Oracle Database SQL functions
 - XMLCOLATTVAL
 - SYS_XMLAGG
 - XMLCDATA
 - XMLROOT
 - Using the DBMS_XMLGEN PL/SQL package

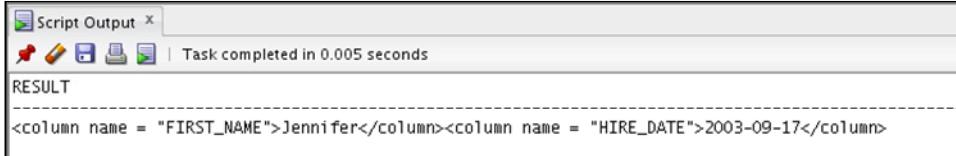


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

XMLCOLATTVAL Function

```
-- XMLCOLATTVAL syntax:  
  
XMLCOLATTVAL (value_expr [, value_expr]...)
```

```
-- The following example generates XML data containing the  
-- first name and hire date of employees who belong to  
-- department 10.  
  
SELECT XMLCOLATTVAL(first_name, hire_date)  
AS result  
FROM employees  
WHERE department_id=10;
```



The screenshot shows the Oracle SQL Developer interface. A script named 'Script Output' is running, indicated by the progress bar at the top which says 'Task completed in 0.005 seconds'. The results section is titled 'RESULT' and contains the generated XML output:

```
<column name = "FIRST_NAME">Jennifer</column><column name = "HIRE_DATE">2003-09-17</column>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

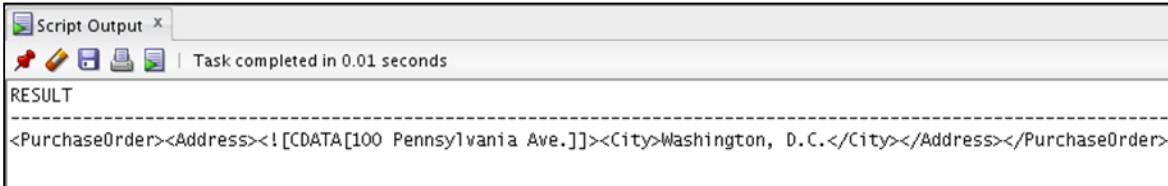
The XMLCOLATTVAL function generates an XML element for each SQL value expression specified in the comma-separated list of arguments. The generated elements have:

- The element tag name as column
- The attribute name, whose value is the argument passed
- The content, which is the value of the argument passed

XMLCDATA Function

```
-- The XMLCDATA function syntax:  
  
XMLCDATA ( value_expr )
```

```
-- The example creates (3) new XML elements and uses the  
-- XMLCDATA function to create an XML CDATA section.  
  
SELECT XMLEMENT("PurchaseOrder",  
                 XMLEMENT("Address",  
                           XMLCDATA('100 Pennsylvania Ave.'),  
                           XMLEMENT("City", 'Washington, D.C.')))  
  
AS RESULT  
FROM DUAL;
```



The screenshot shows the Oracle SQL Developer interface. The 'Script Output' tab is selected, displaying the results of a SQL query. The output window has a toolbar with icons for script, edit, run, and save. Below the toolbar, it says 'Task completed in 0.01 seconds'. The results are labeled 'RESULT' and show the generated XML code:

```
<PurchaseOrder><Address><! [CDATA[100 Pennsylvania Ave.]]><City>Washington, D.C.</City></Address></PurchaseOrder>
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLCDATA function is used to create an XML CDATA section. CDATA sections are useful for writing XML code as text data in an XML document. A CDATA section starts with the sequence `<! [CDATA[` and ends with the first occurrence of the sequence `]]>`. All characters enclosed with these two sequences are interpreted as characters.

The XMLCDATA function returns an instance of XMLType. The function raises an error if the constructed XML is not a legal XML CDATA section. It has the following conditions:

- The value expression must not contain the substring `]]>`.
- If the evaluated result of the value expression is NULL, the function returns NULL.

XMLROOT Function

```
-- The XMLROOT function syntax:
```

```
XMLROOT (value_expr,  
         VERSION {value_expr | NOVALUE}  
         [, STANDALONE {YES | NO | NOVALUE} ] )
```

```
-- Use the XMLROOT function to specify the prolog values of  
-- VERSION and STANDALONE.
```

```
SELECT XMLROOT(XMLType('<po_id>143598</po_id>'),  
               VERSION '1.0', STANDALONE YES)  
AS "XMLROOT"  
FROM DUAL;
```

The screenshot shows the Oracle SQL Developer interface. A script window titled 'Script Output' displays the results of the executed query. The output shows the XML declaration and the value of the XML document. The output window has a toolbar at the top with icons for copy, paste, and refresh, and a status bar at the bottom indicating 'Task completed in 0.009 seconds'.

```
Script Output X  
✖️ ⚙️ 🖊️ 📁 🗃️ 📈 | Task completed in 0.009 seconds  
XMLROOT  
-----  
<?xml version="1.0" standalone="yes"?>  
<po_id>143598</po_id>
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The XMLROOT function is used to create the declaration statement of an XML document. The XMLEMENT function does not generate the XML information (prolog) at the top of the document. You can use the XMLROOT function to specify the prolog values. The XMLROOT function adds a VERSION property and (optionally) a STANDALONE property to the root information of an XML value.

The prolog value STANDALONE indicates whether or not the XML document references an external entity or an external data type specification. The value returned takes the following form: <?xml version = "version" [STANDALONE = "{yes | no}"]?>

- STANDALONE = YES specifies that the XML document *does not* reference an external entity.
- STANDALONE = NO specifies that the XML document refers to one or more external entities.

The example in the slide shows how to specify the prolog values of VERSION and STANDALONE. In the example, the value of VERSION is specified as 1.0. If you specify NOVALUE for VERSION, the VERSION defaults to 1.0. The optional STANDALONE clause is specified as YES. If you omit the STANDALONE clause or if you specify it with NOVALUE, the STANDALONE property is absent from the value returned by the function.

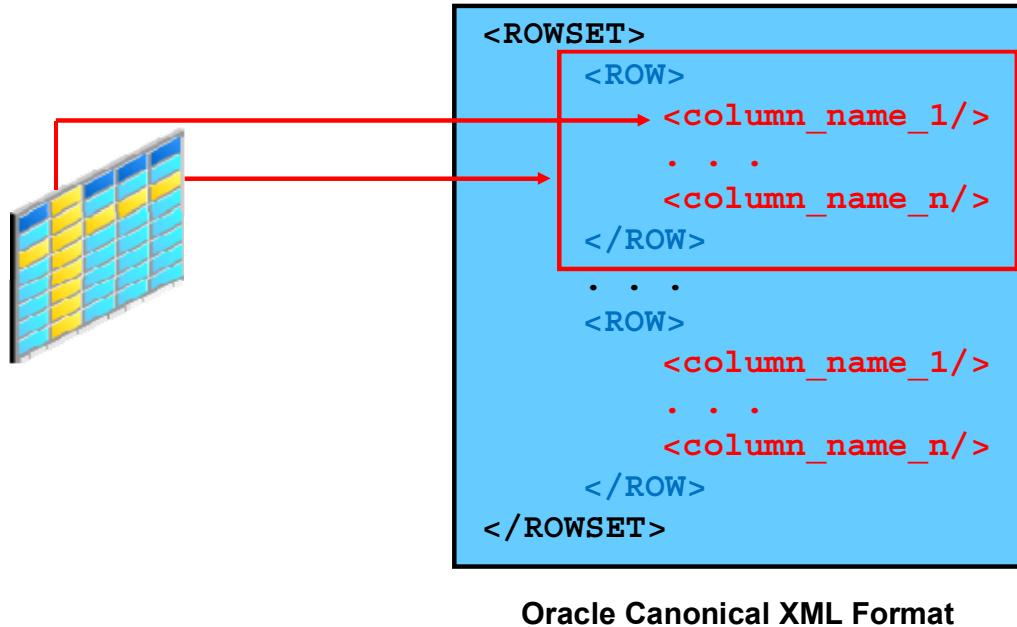
Lesson Agenda

- Generating XML data by:
 - Using XQuery
 - Using the SQL/XML standard functions
 - Using the Oracle Database SQL functions
 - XMLCOLATTVAL
 - SYS_XMLAGG
 - XMLCDATA
 - XMLROOT
 - Using the DBMS_XMLGEN PL/SQL package



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

DBMS_XMLGEN PL/SQL Package



Oracle Canonical XML Format

ORACLE

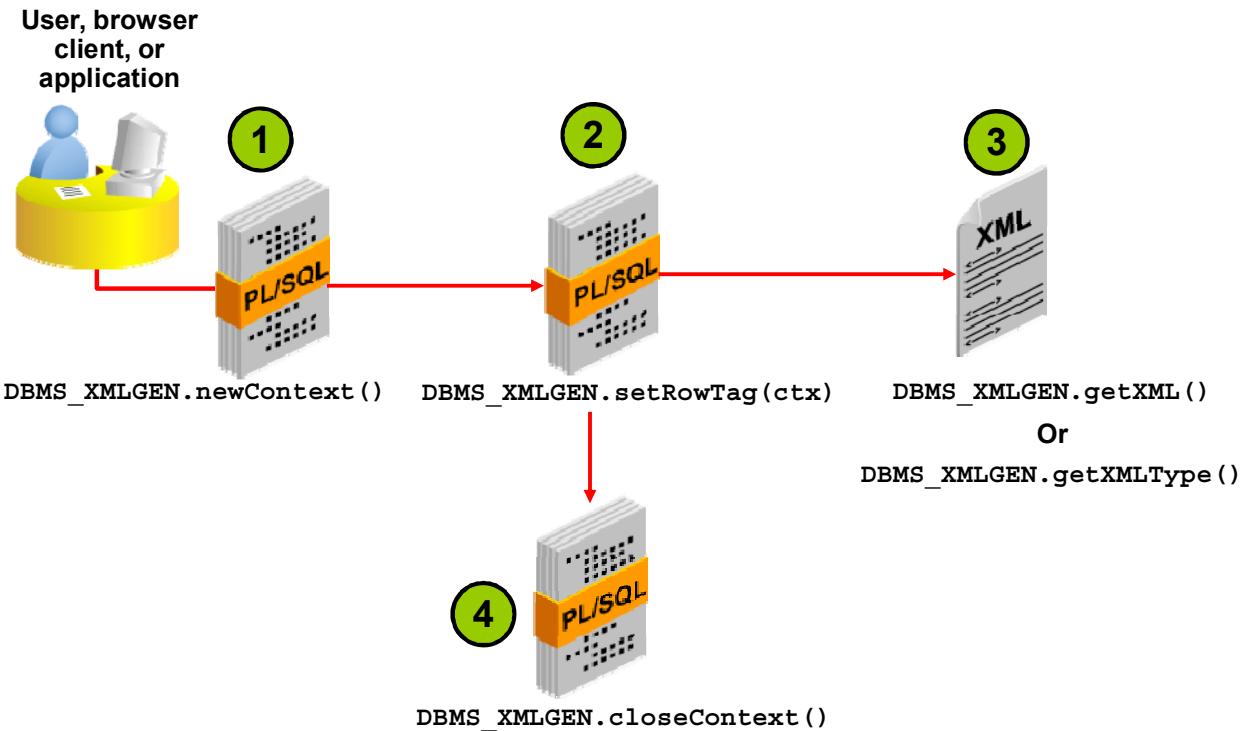
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The DBMS_XMLGEN PL/SQL package converts the results of a SQL query to the Oracle canonical XML format as shown in the slide:

- Each row returned by the SQL query maps to an XML element with the default element name ROW.
- Each column returned by the SQL query maps to a child element of the ROW element.
- The entire result is wrapped in a ROWSET element. The element names ROW and ROWSET can be replaced with names that you choose by using the DBMS_XMLGEN procedures `setRowTagName` and `setRowSetTagName`, respectively.

With DBMS_XMLGEN, you can specify a fetch interface to set the maximum number of rows and the number of rows to skip during retrieval. This feature speeds up response time and is especially useful for the pagination requirements in web applications. For example, when generating a page of XML data, you can restrict the number of rows converted to XML; then, in subsequent calls, you can get the next set of rows, and so on.

Using the DBMS_XMLGEN PL/SQL Package: Steps



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide shows how to use the DBMS_XMLGEN PL/SQL package.

1. Get the context handle from the PL/SQL package by supplying a SQL query and calling the `newContext` function.
2. Pass the context to all the procedures or functions in the PL/SQL package to set the various options. For example, to set the `ROW` element name, use `setRowTag(ctx)`, where `ctx` is the context obtained from the previous `newContext` call.
3. Get the XML result by using `getXML` or `getXMLType`. The `getXML` function returns XML data as a Character Large Object (CLOB) value, and the `getXMLType` function returns XML data as an instance of `XMLType`. By using the `setMaxRows` call to set the maximum rows to be retrieved for each fetch, you can call either of these functions repeatedly, retrieving the maximum number of rows set for each call. These functions return `NULL` if there are no rows left in the query. You can use the `getNumRowsProcessed` function to determine how many rows are retrieved. Reset the query to start again and repeat step 3.
4. Close the context with `closeContext` to release all the resources associated with the context.

Generating Simple XML: Example

```

CREATE TABLE temp_clob_tab(result CLOB);
DECLARE
  qryCtx DBMS_XMLGEN.ctxHandle;
  result CLOB;
BEGIN
  qryCtx := DBMS_XMLGEN.newContext(
    'SELECT * FROM hr.employees');
  DBMS_XMLGEN.setMaxRows(qryCtx, 2); 1
  LOOP
    result := DBMS_XMLGEN.getXML(qryCtx); 2
    EXIT WHEN DBMS_XMLGEN.getNumRowsProcessed(qryCtx) 3
      = 0;
    INSERT INTO temp_clob_tab VALUES(result);
4
  END LOOP;
  DBMS_XMLGEN.closeContext(qryCtx);
END;

```

```

table TEMP_CLOB_TAB created.
anonymous block completed

```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In your application, instead of generating all the XML data for all the rows simultaneously, you may want to retrieve only a few rows. You can use the *fetch* interface of DBMS_XMLGEN to retrieve a fixed number of rows each time. This facilitates faster response time.

The slide example shows how to retrieve a fixed number of rows from the EMPLOYEES table by using the *fetch* interface of DBMS_XMLGEN. The temp_clob_tab table is a table to hold the results as CLOB.

Perform the following steps to retrieve a fixed number of rows from the EMPLOYEES table:

1. Create a new context with the SQL query.
2. Specify the maximum number of rows to get as 2 for each call to *getXML*.
3. Get the XML document by fetching the maximum number of rows specified (2).
4. Insert the results in the temp_clob_tab table.
5. Close the context.

Instead of storing the results in the temp_clob_tab table, you can also print the LOB data or output it to a stream.

Generating Simple XML: Example

```
SELECT *
FROM temp_clob_tab
WHERE rownum < 3;
```



Script Output x | Task completed in 0.035 seconds

RESULT
<?xml version="1.0"?> <ROWSET> <ROW> <EMPLOYEE_ID>124</EMPLOYEE_ID> <FIRST_NAME>Kevin</FIRST_NAME> <LAST_NAME>Mourgos</LAST_NAME> <EMAIL>KMOURGOS</EMAIL> <PHONE_NUMBER>650.123.5234</PHONE_NUMBER> <HIRE_DATE>16-NOV-07</HIRE_DATE> <JOB_ID>ST_MAN</JOB_ID> <SALARY>5800</SALARY> <MANAGER_ID>100</MANAGER_ID> <DEPARTMENT_ID>50</DEPARTMENT_ID> </ROW> <ROW> <EMPLOYEE_ID>125</EMPLOYEE_ID> <FIRST_NAME>Julia</FIRST_NAME> <LAST_NAME>Nayer</LAST_NAME> <EMAIL>JNAYER</EMAIL> <PHONE_NUMBER>650.124.1214</PHONE_NUMBER> <HIRE_DATE>16-JUL-05</HIRE_DATE> <JOB_ID>ST_CLERK</JOB_ID> <SALARY>3200</SALARY> <MANAGER_ID>120</MANAGER_ID> <DEPARTMENT_ID>50</DEPARTMENT_ID> </ROW>

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can query the `temp_clob_tab` table to view the results as shown in the slide.

Generating Recursive XML with a Hierarchical Query

```

CREATE TABLE new_tab(x XMLType);
DECLARE
  qryctx DBMS_XMLGEN.ctxhandle;
  result CLOB;
BEGIN
  qryctx := DBMS_XMLGEN.newcontextFromHierarchy(
    'SELECT level, XMLElement("NAME", last_name) AS myname
     FROM hr.employees
     CONNECT BY PRIOR employee_id=manager_id
     START WITH employee_id = 102');
  DBMS_XMLGEN.setRowSetTag(qryctx, 'mynum_hierarchy');
  result:=DBMS_XMLGEN.getxml(qryctx);
  INSERT INTO new_tab VALUES(XMLType(result));
  COMMIT;
  DBMS_XMLGEN.closecontext(qryctx);
END;
/

```

table NEW_TAB created.
anonymous block completed



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

`DBMS_XMLGEN.newContextFromHierarchy` takes as its argument a hierarchical query string, which is typically formulated with a `CONNECT BY` clause. It returns a context that can be used to generate a hierarchical XML document with recursive elements.

The hierarchical query returns two columns as:

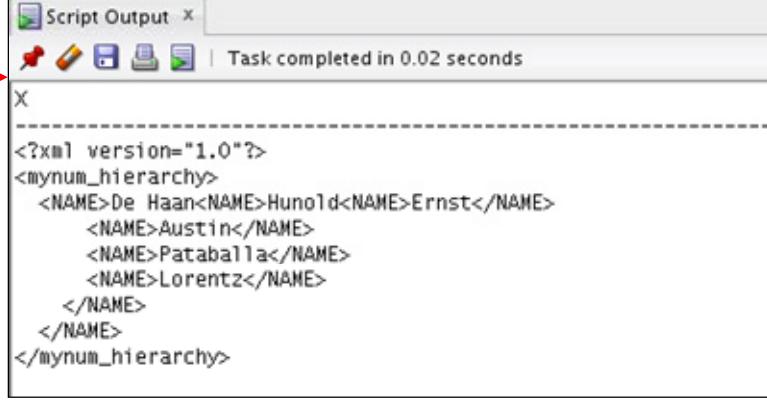
- The level number, which is a pseudocolumn generated by the `CONNECT BY` query
- An `XMLType` column

The example in the slide shows how to generate a manager-employee hierarchy by using `DBMS_XMLGEN.newContextFromHierarchy`.

- `CONNECT BY` specifies the relationship between the parent rows and the child rows of the hierarchy.
- To find the children of the parent row, Oracle XML DB evaluates the `PRIOR` expression of the `CONNECT BY` condition for the parent row (`employee_id=manager_id`).
- The `START WITH` clause specifies the root rows of the hierarchy.
- The `ROWSET` tag is explicitly set by using the `setRowsetTag` procedure.

Generating Recursive XML with a Hierarchical Query

```
SELECT *
FROM new_tab;
```



A screenshot of the Oracle SQL Developer interface. A red arrow points from the code block above to the 'Script Output' window. The window title is 'Script Output' and it shows the results of the query. The output is a single line of XML:

```
<?xml version="1.0"?>
<mynum_hierarchy>
  <NAME>De Haan</NAME><NAME>Hunold</NAME><NAME>Ernst</NAME>
    <NAME>Austin</NAME>
    <NAME>Pataballa</NAME>
    <NAME>Lorentz</NAME>
  </NAME>
</NAME>
</mynum_hierarchy>
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can query the `new_tab` table to view the results as shown in the slide.

Quiz

Which of the following can you use to generate XML data?

- a. XQuery
- b. Standard SQL/XML functions to generate XML
- c. Oracle SQL functions to generate XML
- d. The DBMS_XMLGEN PL/SQL package to generate XML



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a, b, c, d

Summary

In this lesson, you should have learned how to:

- Use XQuery to generate XML
- Use the standard SQL/XML functions to generate XML
- Use the Oracle SQL functions to generate XML
- Use the DBMS_XMLGEN PL/SQL package to generate XML



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 12: Overview

This practice covers the following topics:

- Using XQuery to generate XML
- Using the standard SQL/XML functions to generate XML
- Using the Oracle Database SQL functions to generate XML
- Using the DBMS_XMLGEN PL/SQL package to generate XML



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The goal of this practice is to gain hands-on experience of the functions and procedures that were covered in this section.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only