

Oracle Database 12c: Use XML DB

Student Guide - Volume II

D81146GC10
Edition 1.0
October 2013
D84301

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

13

Transforming and Validating XMLType Data

ORACLE®

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

Objectives

After completing this lesson, you should be able to transform and validate XML data.



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

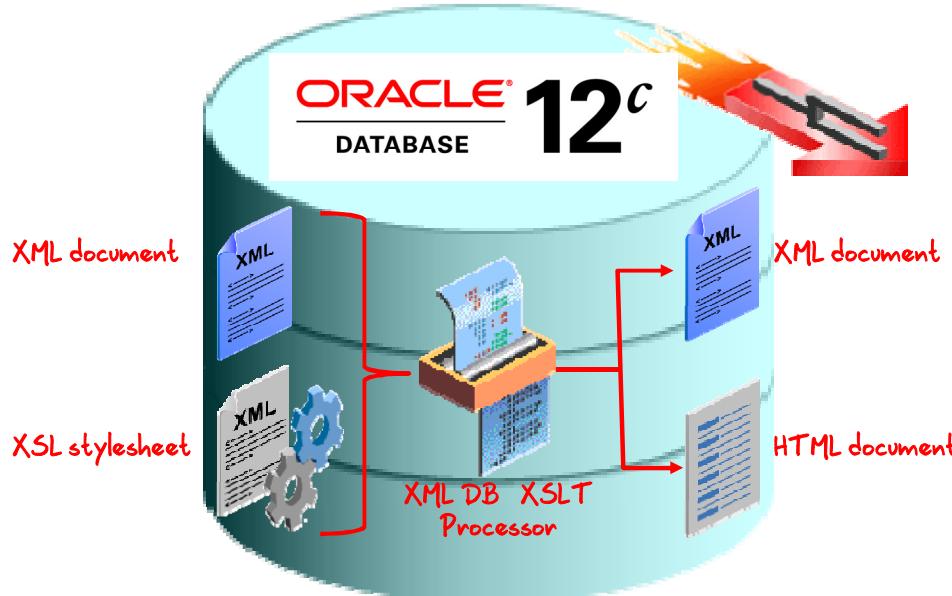
Lesson Agenda

- Transforming XML
- Creating XMLType views
- Validating XMLType instances



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

XSL Transformation and Oracle XML DB



ORACLE

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

The W3C XSLT Recommendation defines an XML language for specifying how to transform XML documents from one form to another. Transformation can include mapping from one XML schema to another or mapping from XML to some other format such as HTML.

XSL transformation is typically expensive in terms of the amount of memory and processing required. Both the source document and the style sheet must be parsed and loaded into memory structures that allow random access to different parts of the documents. Most XSL processors use DOM to provide the dynamic memory representation of both documents. The XSL processor then applies the style sheet to the source document, generating a third document.

Oracle XML DB includes an XSLT processor that lets XSL transformations be performed inside the database. In this way, Oracle XML DB can provide XML-specific memory optimizations that significantly reduce the memory required to perform the transformation. It can also eliminate overhead associated with parsing the documents. These optimizations are available only when the source for the transformation is a schema-based XML document.

Transformation Functions

Oracle XML provides three ways to invoke the XSLT processor:

- SQL function `XMLTransform`
- `XMLType` method `transform()`
- PL/SQL package `DBMS_XSLPROCESSOR`



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

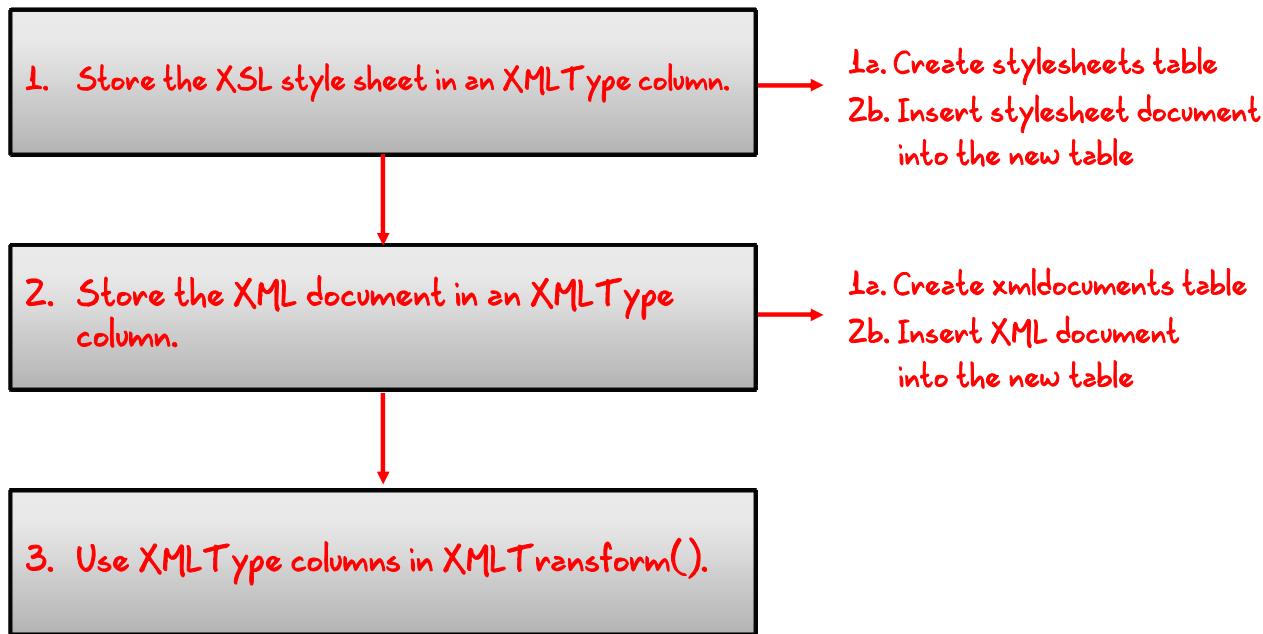
The `XMLTransform()` and `XMLType.transform()` functions transform an XML document into an output document by using the XSL style sheet. Both functions accept two arguments:

1. The XML document as an `XMLType` instance
2. The XSL style sheet as an `XMLType` instance

The transformation functions return the transformed output as an `XMLType` instance. The output can be transformed into another XML document, HTML page, or textual information as specified by the XSL style sheet template rules.

You typically need to use `XMLTransform()` when retrieving and/or generating XML documents stored as `XMLType` in the Oracle Database. You can also use the `XMLTYPE.transform()` function, which produces the same results as the `XMLTransform()` function.

Using the XMLTransform() Function Example: Overview



ORACLE

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

The example overview in the slide shows the required steps to use the XML transformation in the database by using the `XMLTransform()` function. The detailed steps are shown in the next few slides.

Using the XMLTransform() Function, Step 1: Create the XSL Stylesheet Table

```
-- In step 1, create the stylesheets table which has a  
-- NUMBER column and an XMLType column, xsldoc.  
-- The complete code for the INSERT statement is shown  
-- on the next page.  
  
CREATE TABLE stylesheets (id NUMBER(4), xsldoc XMLType);  
  
INSERT INTO stylesheets  
VALUES (1, XMLType('<xsl:stylesheet ...'));
```

table STYLESHEETS created.



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

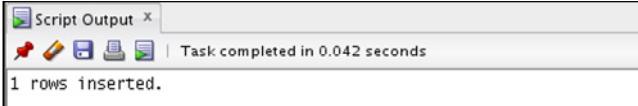
In the first step, you create a table with an XMLType column to store the XSL style sheet that is to be used for transformations. Next, you insert the XSL style sheet into the XMLType column of the stylesheet table. The slide example uses the XMLType() constructor to create an XMLType object that is inserted into the xsldoc column.

Note: The complete code to insert the XSL style sheet into the newly created table is shown in the next slide.

Using the XMLTransform() Function, Step 1: Insert the XSL Stylesheet into the Table

```
-- Insert values in the xsldoc XMLType column of stylesheets

INSERT INTO stylesheets VALUES (1,
XMLType(
'<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html><body>
        <ul><xsl:apply-templates/></ul>
    </body></html>
</xsl:template>
<xsl:template match="employees/employee">
    <li><xsl:value-of select="first_name"/>,
        <xsl:value-of select="last_name"/></li>
</xsl:template>
</xsl:stylesheet>'));
```



ORACLE

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

The slide code example populates the `stylesheets` table with one XSL style sheet document.

Using the XMLTransform() Function: Step 2, Create the XML Documents Table

```
-- In step 2, create the xmldocuments table which has a  
-- NUMBER column and an XMLType column, xmldoc. Next,  
-- store the XML document in the xmldoc XMLType column of  
-- the newly created table.  
-- The complete code for the INSERT statement is shown  
-- on the next page.
```

```
CREATE TABLE xmldocuments (id NUMBER(4), xmldoc XMLType);  
  
-- Complete code on next page  
INSERT INTO xmldocuments  
VALUES (1, XMLType('<employees>...</employees>'));
```

```
table XMLDOCUMENTS created.  
1 rows inserted.
```



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

In the second step, you create a table to store the XML document in an XMLType column by using an INSERT statement with the XMLType() object constructor.

Note: The complete code to insert the XML document into the newly created table is shown in the next slide.

Using the XMLTransform() Function: Step 2, Insert the XML Document

```
-- Next, insert values in the xmldoc XMLType column of the  
-- newly created table,  
  
INSERT INTO xmldocuments VALUES (1,  
XMLType(  
'<?xml version="1.0"?>  
<employees>  
    <employee>  
        <employee_id>100</employee_id>  
        <first_name>Steve</first_name>  
        <last_name>King</last_name>  
    </employee>  
    <employee>  
        <employee_id>101</employee_id>  
        <first_name>Neena</first_name>  
        <last_name>Kocchar</last_name>  
    </employee>  
</employees>'));
```



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

The code example in the slide populates the `xmldocuments` table and with one XML document.

Using the XMLTransform() Function, Step 3: Use XMLType columns in XMLTransform()

```
-- In step 3, Use the XMLTransform() function to
-- transform the XML document with id 1 in the
-- xmldocuments table by using the XSL style sheet with
-- id 1 from the stylesheets table.

SELECT XMLTransform(xmldoc, xsldoc) result
FROM xmldocuments d, stylesheets s
WHERE d.id = s.id
```

The screenshot shows the Oracle SQL Developer interface. The 'Script Output' tab is selected, displaying the results of a query. The output is titled 'RESULT' and contains the following XML output:

```
<html>
<body>
<ul>
<li>Steve,
    King</li>
<li>Neena,
    Kocchar</li>
</ul>
</body>
</html>
```

ORACLE

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

In the third step, you execute a query by using the XMLTransform() function to transform the XML document with id 1 in the xmldocuments table by using the XSL style sheet with id 1 from the stylesheets table. The query can also be written as follows:

```
SELECT XMLType.transform(xmldoc,
  (SELECT xsldoc FROM stylesheets WHERE id = 1)) result
FROM xmldocuments d
WHERE d.id = 1
```

```
<html>
<body>
<ul>
<li>Steve,
    King</li>
<li>Neena,
    Kocchar</li>
</ul>
</body>
</html>
```

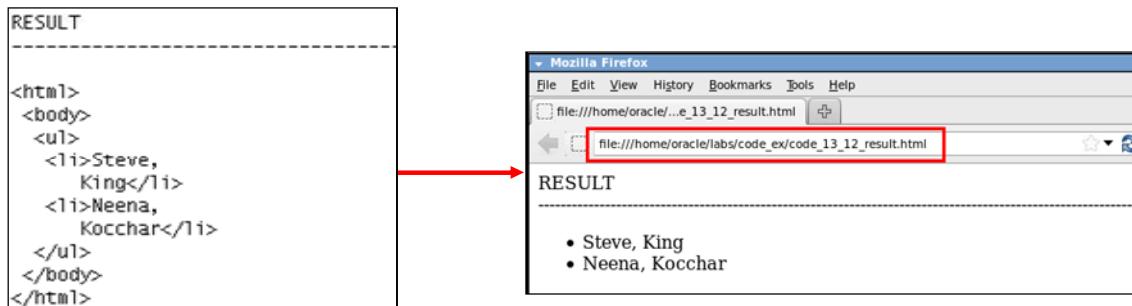
Note: The XMLType.transform() function can be used instead of the XMLTransform() function. The complete code example is shown in the next slide.

Using the XMLTransform() Function

- Write the transformation query:

```
-- Query from previous page.
SELECT XMLTransform(xmldoc, (SELECT xsldoc
                               FROM stylesheets
                               WHERE id = 1)) result
FROM xmldocuments d
WHERE d.id = 1;
```

- Spool (or copy) these results to an HTML file for viewing in a browser:



ORACLE

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

The example in the earlier slides used the XSL style sheet to transform an XML document into an HTML page.

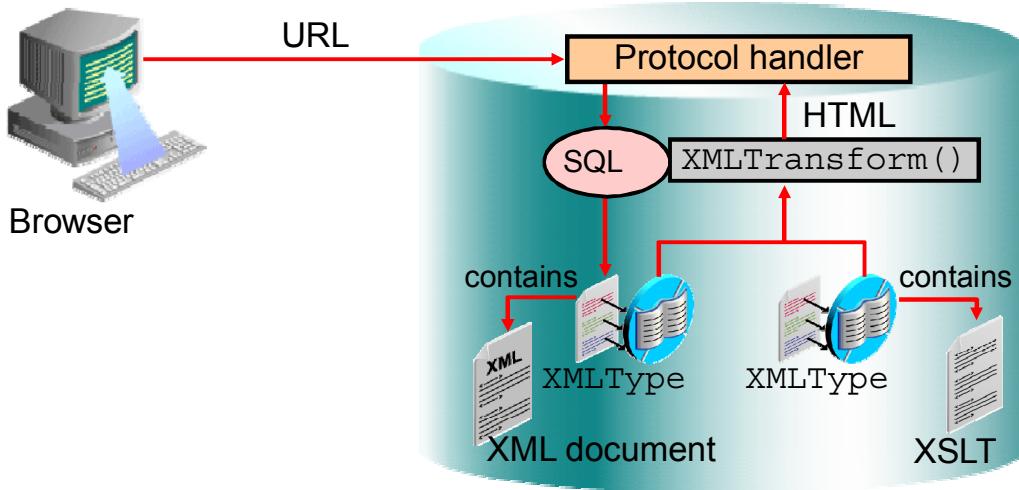
The HTML results can be copied and pasted into a text editor and saved, or spooled from SQL*Plus, into a file with an .htm or .html extension.

The HTML file can be viewed by using Internet Explorer or any other browser.

Note: You can also open the HTML file in JDeveloper and use the HTML Editor Preview tab to observe the final result.

Benefits of XML Transformation

Using Oracle XML DB transformation functions provides better performance when processing XML stored in the database.



ORACLE

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

An XML document describes the content and does not provide presentation information. Applying XSLT to an XML document can transform the XML content into a specific presentation format. Oracle XML DB supports the W3C XSLT standard by providing support for performing XSLT transformations inside the database. Because the transformation takes place close to the data, Oracle XML DB can optimize features, such as memory usage, I/O operations, and the network traffic that is required to perform the transformation. Most transformations are from XML to HTML or XML. Oracle XML DB supports transformation to Wireless Markup Language (WML) as well.

Using XDK Versus Oracle XML DB for XML Transformation

Users who are required to perform transformations in the middle tier must use Oracle XDK. Middle tier-based performance distributes processing, but can lead to additional complexity and some performance penalty. Using the XDK is the preferred approach for accessing information that is stored outside the database.

Using the features of Oracle XML DB is the best for processing the XML and XSL data that is stored within the database. Transformation in the database gives better overall performance when the data is stored within the database. However, additional processing adds more load on the database.

Lesson Agenda

- Transforming XML
- Creating XMLType views
- Validating XMLType instances



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

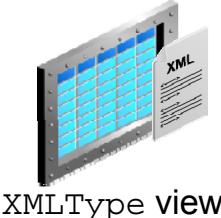
XMLType Views

An XMLType view:

- Wraps existing relational or object-relational data in XML formats
- Shows the data as a collection of XMLType instances

```
-- Syntax to create an XMLType view

CREATE OR REPLACE VIEW view_name
  OF XMLType WITH OBJECT ID {DEFAULT| (expression) }
AS sql_query;
```



ORACLE

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

An XMLType view wraps data stored in the relational or object-relational tables in XML formats. Similar to an object view, an XMLType view presents the contents as XMLType instances. The main advantage of using an XMLType view is the ability to experiment with various forms of storage, apart from the object-relational and binary XML storage choices that are available to the XMLType tables.

An XMLType view is created by using a CREATE OR REPLACE VIEW statement with the OF XMLType WITH OBJECT ID clause, as shown in the slide.

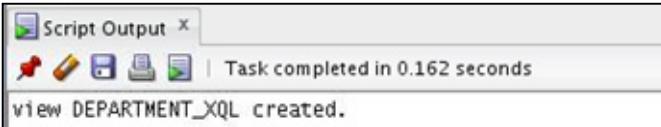
In an Oracle Database, object tables, XMLType tables, object views, and XMLType views do not have column names specified. Therefore, a system-generated pseudocolumn OBJECT_ID can be used in queries. OBJECT_ID uniquely identifies objects in an object view by using the WITH OBJECT ID clause and the DEFAULT keyword, or by specifying an expression to be used as a unique identifier.

Note: There are two types of XMLType views: Non-schema-based XMLType views and XML schema-based XMLType views. For more information about XMLType views, see *Oracle XML DB Developer's Guide 12c Release 1 (12.1)*.

Creating an XMLType View

```
-- The complete code example is on the next full notes pages.

create or replace view DEPARTMENT_XQL of xmldtype
with object id
(XMLCAST (
    XMLQUERY('/Department/@DepartmentId'
    passing OBJECT_VALUE
    returning content)
AS NUMBER))
AS
SELECT COLUMN_VALUE FROM XMLTABLE
('for $d in fn:collection("oradb:/HR/DEPARTMENTS") ,
 $l in fn:collection("oradb:/HR/LOCATIONS") ,
 $c in fn:collection("oradb:/HR/COUNTRIES")
where $d/ROW/LOCATION_ID = $l/ROW/LOCATION_ID
and $l/ROW/COUNTRY_ID = $c/ROW/COUNTRY_ID
return
...
')
```



ORACLE

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

You can create XMLType views by using the XMLQuery SQL function or based on the SQL/XML generation functions, such as XMLEMENT, XMLFOREST, XMLCONCAT, and XMLAGG, and the XMLCOLATTVAL Oracle Database extension function.

The slide example shows that you can use XQuery to generate XML from SQL data by using views. The example creates an XMLType view over the database tables by using the XMLTable function and an XQuery expression.

The following code example uses XQuery to query the XMLType view, which was created by using XQuery over relational tables:

```
SELECT t.column_value
FROM department_xql x,
xmltable('for $i in .
WHERE $i/Department/EmployeeList/Employee/LastName="Grant"
return $i/Department/Name'
passing value(x) ) t;
```

COLUMN_VALUE

<Name>Shipping</Name>

The complete code of the example in the previous slide is as follows:

```
create or replace view DEPARTMENT_XQL of xmldtype
with object id
(XMLCAST (
    XMLQUERY('/Department/@DepartmentId'
    passing OBJECT_VALUE
    returning content)
AS NUMBER))
as
SELECT COLUMN_VALUE FROM XMLTABLE
('for $d in fn:collection("oradb:/HR/DEPARTMENTS") ,
 $l in fn:collection("oradb:/HR/LOCATIONS") ,
 $c in fn:collection("oradb:/HR/COUNTRIES")
where $d/ROW/LOCATION_ID = $l/ROW/LOCATION_ID
and $l/ROW/COUNTRY_ID = $c/ROW/COUNTRY_ID
return
<Department DepartmentId= "{$d/ROW/DEPARTMENT_ID/text()}" >
<Name>{$d/ROW/DEPARTMENT_NAME/text()}</Name>
<Location>
<Address>{$l/ROW/STREET_ADDRESS/text()}</Address>
<City>{$l/ROW/CITY/text()}</City>
<State>{$l/ROW/STATE_PROVINCE/text()}</State>
<Zip>{$l/ROW/POSTAL_CODE/text()}</Zip>
<Country>{$c/ROW/COUNTRY_NAME/text()}</Country>
</Location>
<EmployeeList>
{
for $e in fn:collection("oradb:/HR/EMPLOYEES") ,
 $m in fn:collection("oradb:/HR/EMPLOYEES") ,
 $j in fn:collection("oradb:/HR/JOB$")
where $e/ROW/DEPARTMENT_ID = $d/ROW/DEPARTMENT_ID
and $j/ROW/JOB_ID = $e/ROW/JOB_ID
and $m/ROW/EMPLOYEE_ID = $e/ROW/MANAGER_ID
return
<Employee employeeNumber="{$e/ROW/EMPLOYEE_ID/text()}" >
<FirstName>{$e/ROW/FIRST_NAME/text()}</FirstName>
<LastName>{$e/ROW/LAST_NAME/text()}</LastName>
<EmailAddress>{$e/ROW/EMAIL/text()}</EmailAddress>
<Telephone>{$e/ROW/PHONE_NUMBER/text()}</Telephone>
```

```
<StartDate>{$e/ROW/HIRE_DATE/text()}</StartDate>
<JobTitle>{$j/ROW/JOB_TITLE/text()}</JobTitle>
<Salary>{$e/ROW/SALARY/text()}</Salary>
<Manager>{$m/ROW/LAST_NAME/text(), " ", 
$m/ROW/FIRST_NAME/text()}</Manager>
</Employee>
}
</EmployeeList>
</Department>' );
```

The preceding code creates an `XMLType` view from relational tables by using XQuery expressions along with the `fn:collection` function.

Note: For additional information about creating relational views over XML data, see chapter 18 titled Relational Views over XML Data, in the *Oracle XML DB Developer's Guide 12c Release 1 (12.1)* reference guide.

Lesson Agenda

- Transforming XML
- Creating XMLType views
- Validating XMLType instances



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

Validating XMLType Instances

- XML schema-based data that is stored as binary XML:
 - It is automatically validated fully whenever it is inserted or updated
 - This validation does not require building a DOM. It is done using streaming, which is efficient and minimizes memory use.
- For XMLType data that is stored object-relationally:
 - Full validation requires building a DOM, which can be costly in terms of memory management
 - Oracle XML DB does not automatically perform full validation when you insert or update data that is stored o-r
 - In the process of decomposing XML data to store it o-r, Oracle XML DB does automatically perform partial validation



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

Often, besides knowing whether a particular XML document is well-formed, you need to know whether it conforms to a given XML schema, that is, whether it is valid with respect to that XML schema.

XML schema-based data that is stored as binary XML it is automatically validated fully whenever it is inserted or updated. This validation does not require building a DOM. It is done using streaming, which is efficient and minimizes memory use.

For XMLType data that is stored object-relationally, full validation requires building a DOM, which can be costly in terms of memory management. For this reason, Oracle XML DB does not automatically perform full validation when you insert or update data that is stored object-relationally.

However, in the process of decomposing XML data to store it object-relationally, Oracle XML DB does automatically perform partial validation, to ensure that the structure of the XML document conforms to the SQL data type definitions that were derived from the XML schema.

If you require full validation for XMLType data stored object-relationally, then consider validating on the client before inserting the data into the database or updating it.

Performing Full Validation

SQL Function or XMLType Method	Description
Oracle SQL function XMLIsValid and IsSchemaValid() XMLType method	Run the validation process unconditionally. Do not record any validation status. Return: <ul style="list-style-type: none"> • 1 if the document is determined to be valid • 0 if the document is determined to be invalid or the validity of the document cannot be determined.
SchemaValidate() XMLType method	Runs the validation process if the validation status is 0, which it is by default. Sets the validation status to 1 if the document is determined to be valid. (Otherwise, the status remains 0.)
isSchemaValidated() XMLType method	Returns the recorded validation status of an XMLType instance
setSchemaValidated() XMLType method	Sets (records) the validation status of an XMLType instance.



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

Note that the validation status indicates knowledge of validity, as follows:

- 1 means that the document is known to be valid.
- 0 means that validity of the document is unknown. The document might have been shown to be invalid during a validation check, but that invalidity is not recorded. A recorded validation status of 0 indicates only a lack of knowledge about the document's validity.

Quiz

In Oracle XML DB, XMLType instances of the XML data stored in XMLType tables, columns, or views can be transformed (formatted) into HTML, XML, and other mark-up languages by using the XSL style sheets and the `transform()` XMLType function.

- 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 transform and validate XML data.



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

Practice 13: Overview

This practice covers transforming XML data.



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

14

Working With the Oracle XML DB Repository

ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Describe the key features and architecture of the Oracle XML DB Repository
- Create folders and resources in the repository
- Use the RESOURCE_VIEW and PATH_VIEW APIs to access the repository
- Use access control lists
- Describe compound documents
- Describe repository events



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

Lesson Agenda

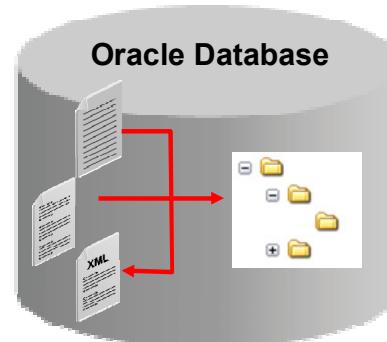
- XML DB Repository: Overview
- Creating folders and resources using PL/SQL
- Accessing resources
 - SQL access
 - Navigational access
- Access control lists
- Compound documents
- Repository events



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

Oracle XML DB Repository: Overview

- The Oracle XML DB Repository is used to organize and manage database content in the form of files and folders called resources.
- Each resource:
 - Is identified by a name and paths
 - Has content (XML or non-XML)
 - Has system-defined (and optional user-defined) metadata
 - Has an access control list
 - Has information stored in the `XMLType` table
`XDB$RESOURCE`

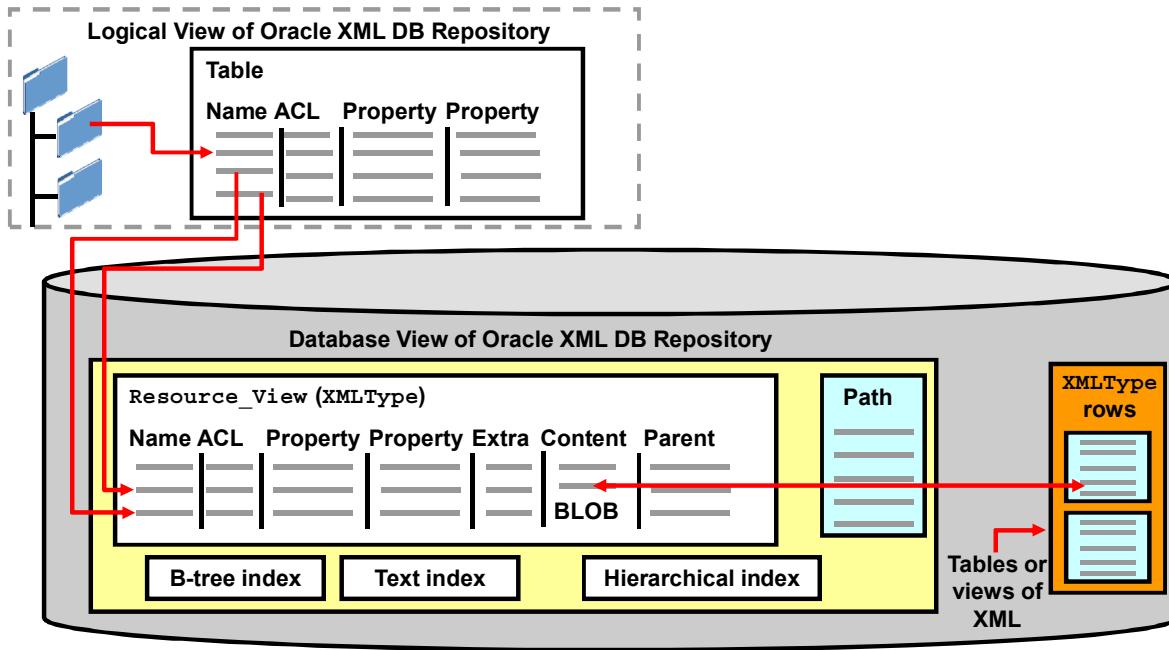


ORACLE®

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

Using Oracle XML DB Repository, you can store content in the database in hierarchical structures, as opposed to traditional relational database structures.

Oracle XML DB Repository: Architecture



ORACLE

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

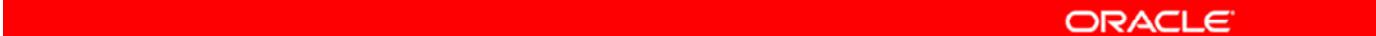
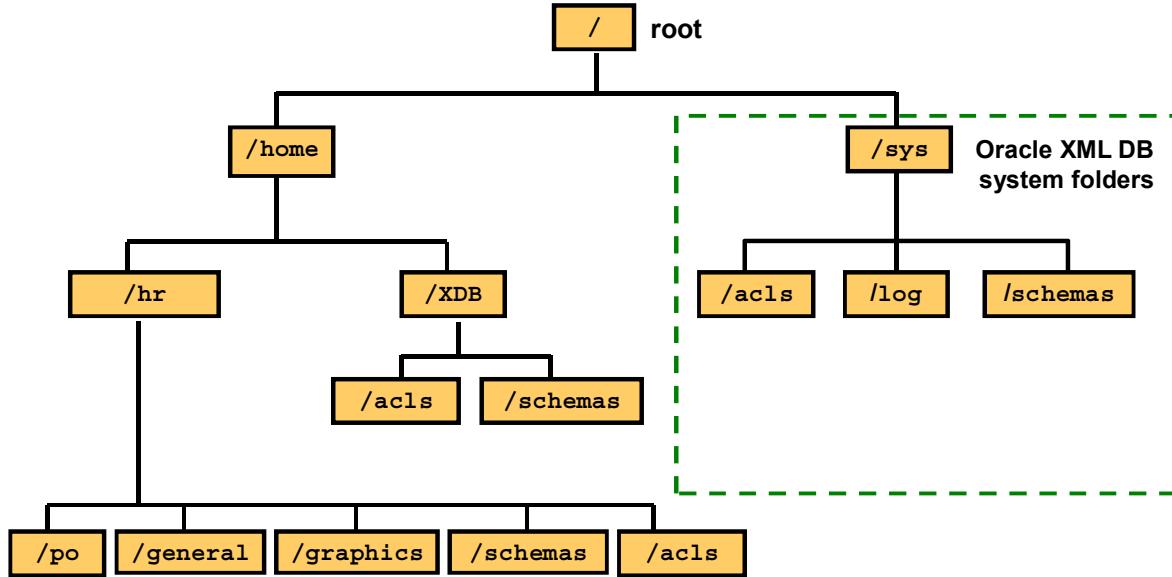
To access the Oracle XML DB Repository, use the RESOURCE_VIEW API. Every resource (content managed by the Oracle XML DB Repository) has a name, an associated ACL (to determine resource accessibility), static properties, and some extra extensible properties. If the content of the resource is XML, RESOURCE_VIEW redirects it to the XMLType row that stores the content. Otherwise, it stores the content as a Binary Large Object (BLOB).

Applications using the repository obtain a logical view of the folders in a parent/child arrangement.

Such a relationship between the folders (necessary to construct the hierarchy) is maintained and traversed by using the hierarchical index. Text indexes are available to search the properties of a resource. Internal B-tree indexes over the Name and ACL attributes speed up access to these attributes of a resource.

In addition to the resource information, RESOURCE_VIEW also contains a Path column, which holds the paths to each resource.

Hierarchical Structures in the Repository



ORACLE

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

The slide shows an example of a typical tree of folders in the Oracle XML DB Repository. At the top of the tree is the root folder “/.” Foldering enables applications to access hierarchically indexed content in the database by using FTP, Hypertext Transmission Protocol, Secure or HTTPS, and Web-based Distributed Authoring and Versioning (WebDAV) protocol standards as if the database content were stored in a file system. Oracle XML DB uses the /sys folder to maintain XML schemas, ACLs, and so on. Whenever you register a schema, Oracle XML DB stores it as a repository resource in the /sys/schemas folder. All the ACLs are stored in the /sys/acls folder. Adding or modifying any data manually in the /sys folder can result in an error.

Links in Oracle XML DB

There are two types of links:

- Repository (folder) links
- Document links

Depending on whether these links target the repository resources, they can be of two types:

1. Hard Links	2. Weak Links
Point to repository resource identifiers	
Resource <i>cannot</i> be deleted if it is the target of a hard link.	Resource <i>can</i> be deleted even if it is the target of a weak link.
It is acyclic.	It can be cyclic.
Provide referential integrity: If a link exists, so does the resource.	



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

The two types of links in Oracle XML DB include document links and repository links:

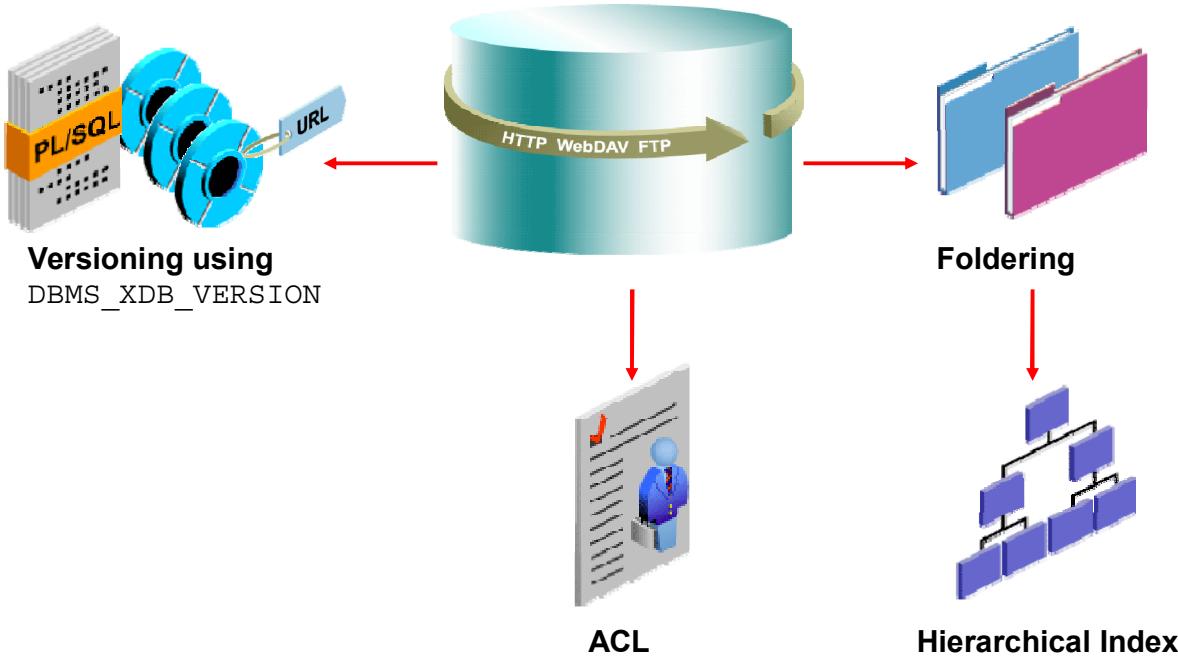
- **Repository links:** A repository folder resource can contain links to other folder or file resources. These links are called repository links. Repository links form the repository's folder-child hierarchical relationships and can be navigated by using the file system–enabled protocols.
- **Document links:** These are the links provided by the XLink and XInclude standards, which are also supported by Oracle XML DB. XLink and XInclude provide links between documents that may reside inside or outside the repository. You learn more about links in the lesson titled "*Oracle XML DB Repository: Advanced Concepts*." For additional information on XLink and XInclude, see the Oracle XML DB Developer's Guide 12c Release 1 (12.1).

Links, if they target the repository resources, can be either hard links or weak links. Both hard and weak links point to the repository resource identifiers. The difference is in whether they allow their target resource to be deleted or not.

- **Hard links:** A resource *cannot be deleted* as long as it remains the target of a hard link. If you delete a hard link, the resource is also deleted if it is not versioned and the deleted hard link is the last hard link to the resource. Hard links to ancestor folders are not permitted, because this would introduce cycles. It is this acyclic nature of hard links that gives the Oracle XML DB Repository its hierarchical structure.
- **Weak links:** A resource *can be deleted* even if it is the target of a weak link (as long as it is not the target of a hard link). If you delete the resource that is the target of one or more weak links, the weak links are also automatically deleted. Weak links can target any folder, possibly creating a cycle.

Both hard and weak links provide referential integrity. If a link exists, so does the resource.

Oracle XML DB Repository Services



ORACLE

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

In addition to supporting APIs that access and manipulate data, the Oracle XML DB Repository provides APIs for the following services:

Oracle XML DB Versioning

The Oracle XML DB Repository allows different versions of the same resource to be maintained. Subsequent updates to the resource (such as adding a table or column) result in new versions being created while data corresponding to the previous versions is retained. The `DBMS_XDB_VERSION` PL/SQL package is used for versioning resources in the Oracle XML DB Repository.

Oracle XML DB ACL Security

For every request to access a resource, the resource's ACL (which contains its privileges) is queried to determine whether the requested operation is legal or not. An ACL is an XML document that contains a set of access control entries (ACEs) for the resources in the repository. Each ACE grants or revokes a set of permissions to a particular user or group (database role).

Oracle XML DB Foldering

Oracle XML DB foldering enables you to store content in the database in a hierarchical structure, as opposed to traditional relational database table structures. The foldering module manages a persistent hierarchy of containers (folders or directories) and resources. Other Oracle XML DB modules—such as protocol servers, the schema manager, and the Oracle XML DB `RESOURCE_VIEW` API—use the foldering module to map path names to resources.

Hierarchical Index

Oracle XML DB uses a hierarchical index to speed up folder-based and path-based traversal. An Oracle-proprietary hierarchical index enables the database to resolve folder-restricted queries without relying on a `CONNECT BY` operation. As a result, Oracle XML DB can execute path-based and folder-restricted queries efficiently.

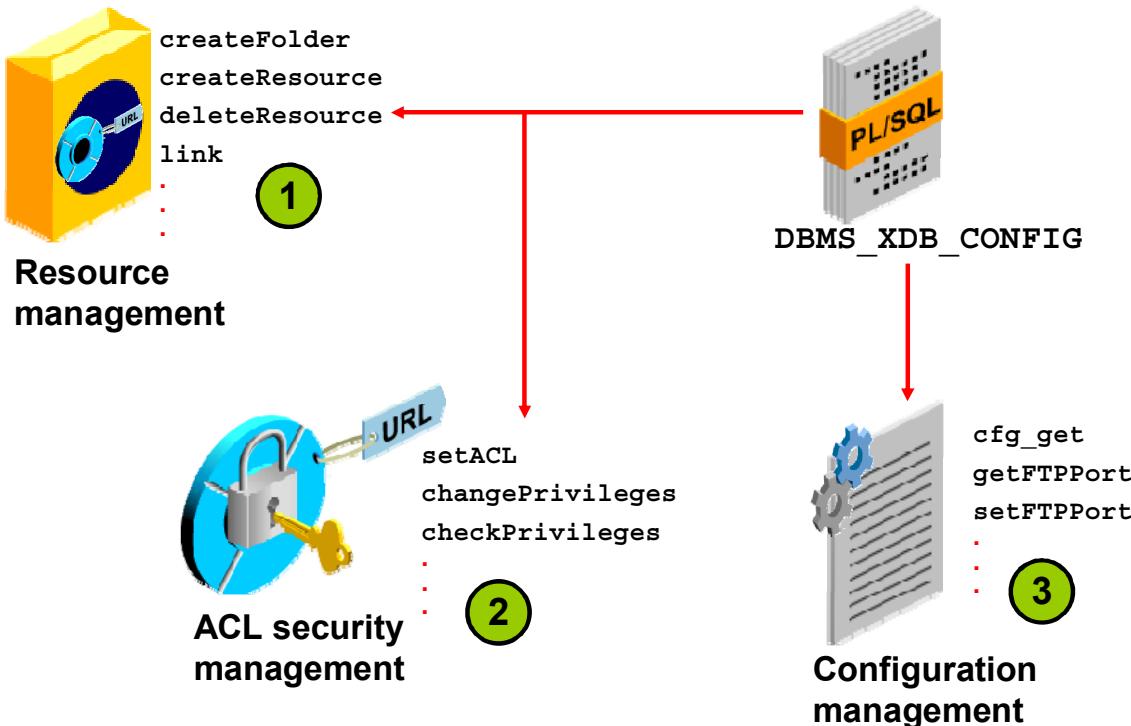
Lesson Agenda

- XML DB Repository: overview
- Creating folders and resources using PL/SQL
- Accessing resources
 - SQL access
 - Navigational access
- Access control lists
- Compound documents
- Repository events



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

Oracle XML DB Resource API for PL/SQL (DBMS_XDB_CONFIG)



ORACLE

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

DBMS_XDB_CONFIG is the Oracle XML DB Resource API for PL/SQL (also known as the PL/SQL Foldering API). You can use the PL/SQL functions and procedures that are available in the PL/SQL DBMS_XDB_CONFIG package to manage the following:

- The Oracle XML DB resources
- The Oracle XML DB security based on ACLs
- The Oracle XML DB configuration

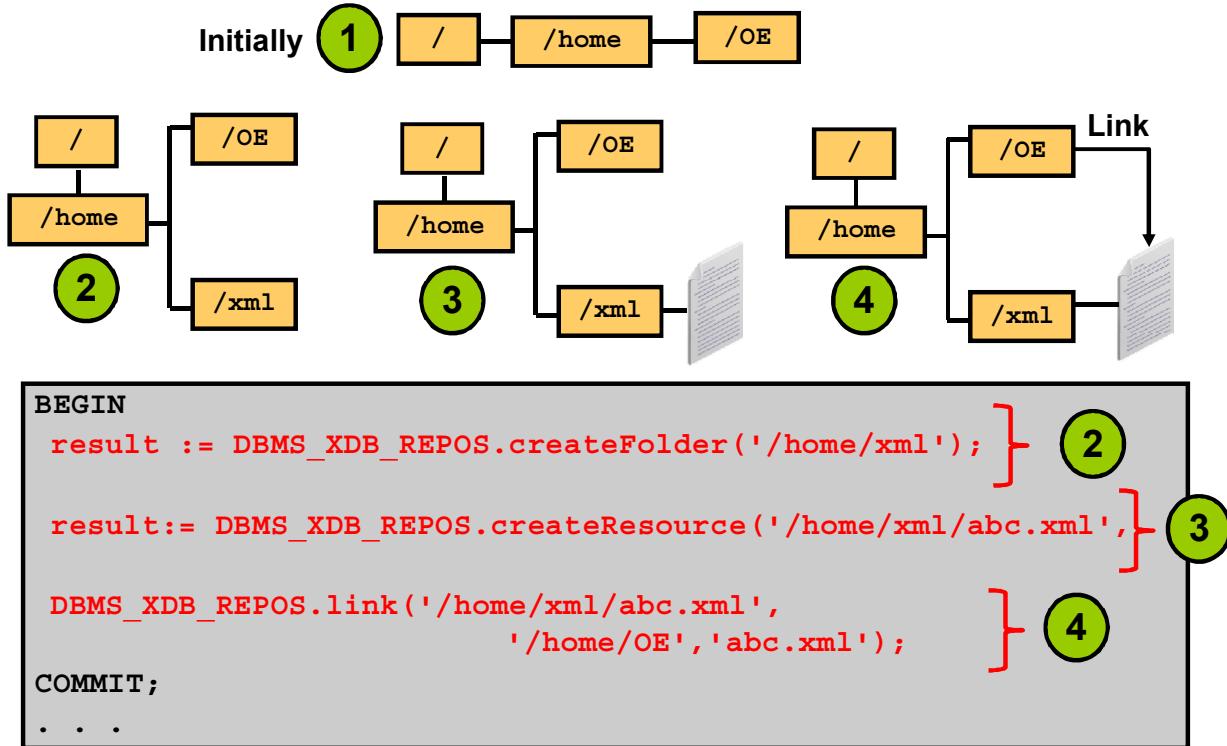
Some of the PL/SQL functions and procedures that are available are listed in the slide.

Note: PL/SQL package DBMS_XDB_CONFIG is new. All Oracle XML DB configuration functions, procedures, and constants have been moved from package DBMS_XDB to DBMS_XDB_CONFIG. They are deprecated for package DBMS_XDB. Oracle recommends that you use them in package DBMS_XDB_CONFIG instead.

PL/SQL Package DBMS_XDB Is Split. The subprograms and constants of PL/SQL package DBMS_XDB have been divided among the following packages: DBMS_XDB_ADMIN, DBMS_XDB_CONFIG, and DBMS_XDB_REPO.

Packages DBMS_XDB_CONFIG and DBMS_XDB_REPO are new in Oracle Database 12c Release 1 (12.1.0.1). Package DBMS_XDB continues to exist for backward compatibility.

Creating Folders and Resources by Using PL/SQL



ORACLE

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

The example in the slide shows how to create folder and file resources in the repository.

1. Initially, the repository has only one folder called `OE` in the `home` folder.
2. Create an `xml` folder in `home` by using `DBMS_XDB_REPO.createFolder`.
3. Create the `abc.xml` resource by using `DBMS_XDB_REPO.createResource`.
4. Link the resource to the `/home/OE` folder by using `DBMS_XDB_REPO.link`.

If the path does not exist, `createFolder` returns an error. You can also use the syntax on the next page to verify the existence of the folders before creating them.

Creating Folders and Resources by Using PL/SQL

```

DECLARE
    result BOOLEAN;
    xmldoc VARCHAR2(250);
BEGIN
    xmldoc := '<EMPLOYEE>
        <ENAME>Locke</ENAME>
        <SALARY>7000</SALARY>
    </EMPLOYEE>';
    IF (not DBMS_XDB_REPO$existsResource('/home/xml')) then
        result := DBMS_XDB_REPO$.createFolder('/home/xml');
    END IF;
    IF (not DBMS_XDB_REPO$.existsResource('/home/xml/abc.xml'))
        then
        result:= DBMS_XDB_REPO$.createResource('/home/xml/abc.xml',
            XMLDOC);
    END IF;
    DBMS_XDB_REPO$.link('/home/xml/abc.xml', '/home/OE', 'abc.xml
    ');
COMMIT;
END;

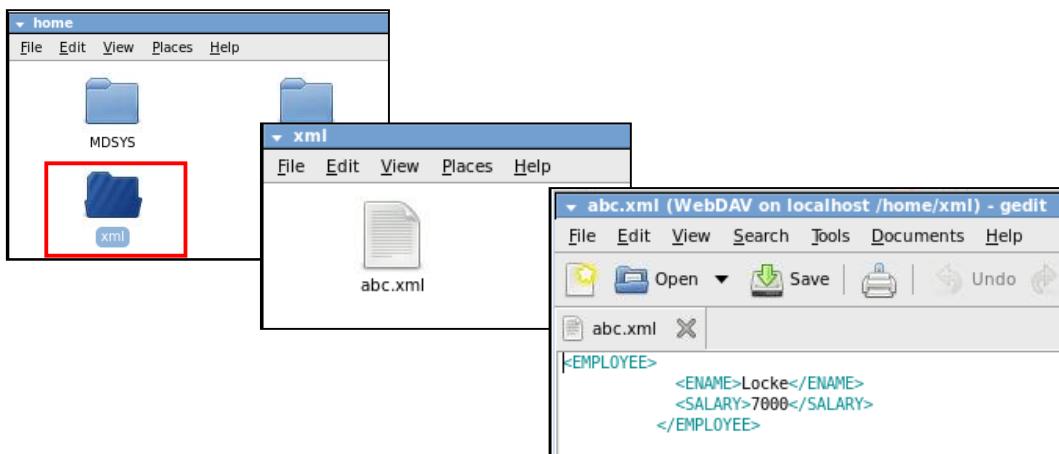
```



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

In the code example on the previous page, if the path does not exist, `createFolder` returns an error. You can also use the code example in the slide to verify the existence of the folders before creating them.

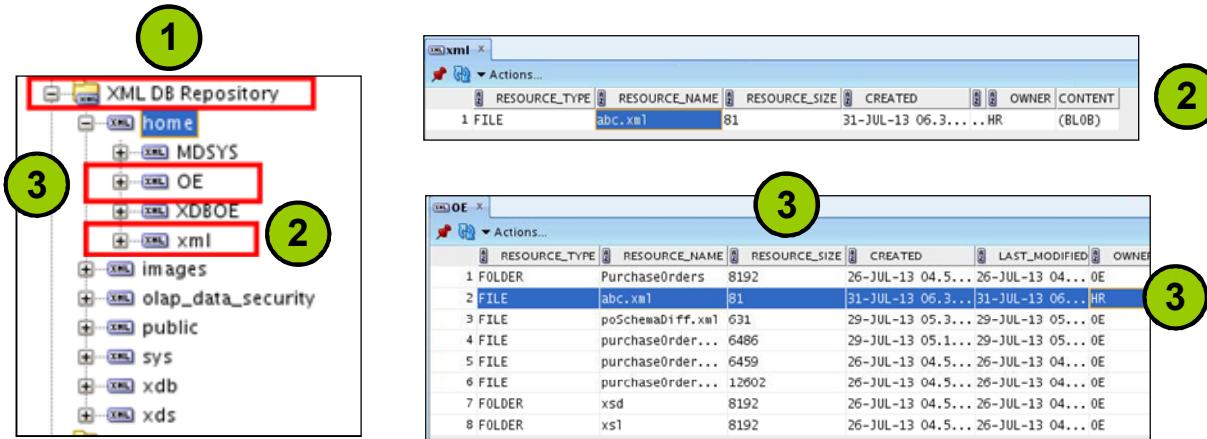
The result of the slide example accessed using the WebDAV connection is as follows:



Note: You cannot visualize the folders in FTP, HTTP, or WebDAV until they are committed.

Viewing Folders and Resources in SQL Developer

```
BEGIN
    result := DBMS_XDB_REPOS.createFolder('/home/xml');
}
result:= DBMS_XDB.createResource('/home/xml/abc.xml',
}
DBMS_XDB.link('/home/xml/abc.xml',
    '/home/oe', 'emp_abc.xml');
```



ORACLE

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

To view XML DB Repository resources in SQL Developer, expand a schema node and click an XML DB Repository object. The screenshots in the slide show folder and file resources that you created in the previous slide in the repository.

1. This image displays the `xml` folder in the repository folder `home`.
2. When you click the `/home/xml` folder, SQL Developer displays its contents. The image displays the `abc.xml` resource that you created by using `DBMS_XDB_REPOS.createResource`.
3. When you click the `/home/oe` folder, SQL Developer displays its `abc.xml` as one of its contents. Because `abc.xml` is a link, `HR` is displayed as the owner of this resource.

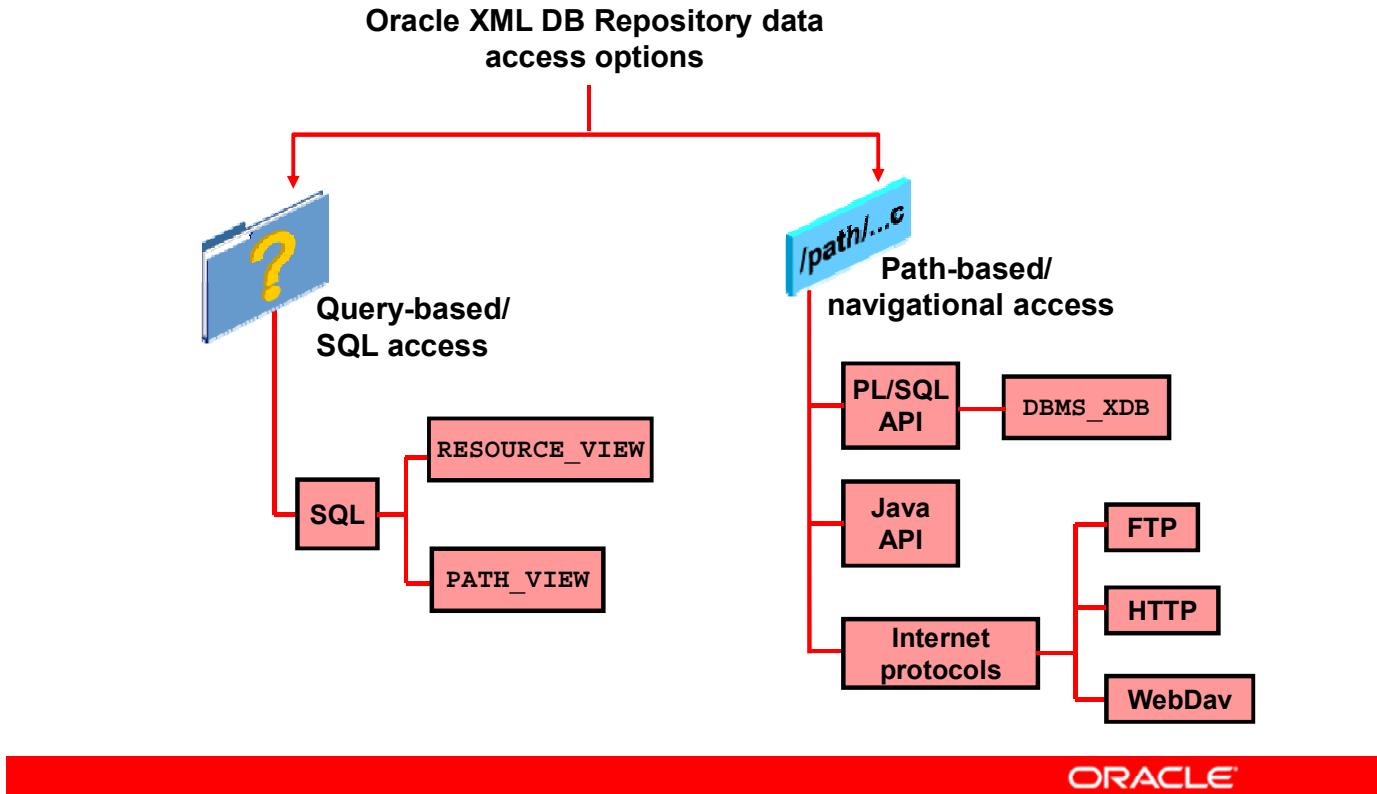
Lesson Agenda

- XML DB Repository: overview
- Creating folders and resources using PL/SQL
- Accessing resources
 - SQL access
 - Navigational access
- Access control lists
- Compound documents
- Repository events



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

Accessing Resources



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

Resource Management: Accessing Resources

The Oracle XML DB Repository provides two ways to access resources: the SQL path and the navigational path.

SQL Access

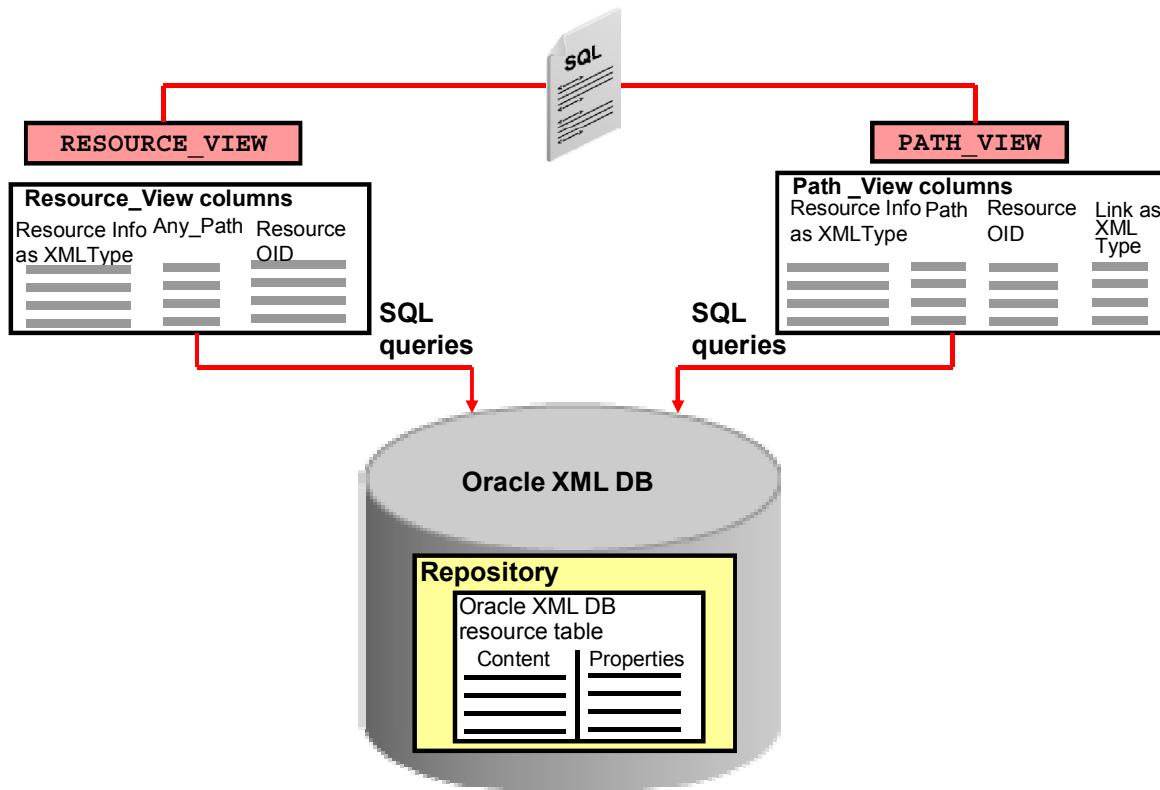
SQL access to the repository is accomplished through a set of views that expose resource properties, path names, and hierarchical access operators to the Oracle XML DB schema. Two views enable SQL access to the Oracle XML DB Repository data:

- RESOURCE_VIEW
- PATH_VIEW

Navigational Access

Navigational access bypasses the SQL engine to provide fast data retrieval. However, because the path name is the only way to retrieve the object, navigational access is restricted. Path-based access is achieved by using a hierarchical index of objects or resources. Each resource has one or more unique path names that reflect its location in the hierarchy. You can use navigational access to reference any `XMLType` object in the database, regardless of its location in the relational tablespace. Navigational access is available through programmatic interfaces in Java, PL/SQL, and C. In this course, you learn to access the repository by using the Internet protocols.

Accessing Resources by Using SQL Access



ORACLE

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

RESOURCE_VIEW and **PATH_VIEW** provide a SQL mechanism to access data that is stored in the repository by using protocols such as FTP and WebDAV, or by using APIs.

RESOURCE_VIEW:

- Is created and defined in `catxdbr.sql`
- Displays only one of the possible path names to the resource, with each row displaying only one resource
- Is widely applicable because many Internet applications need only one URL to access a resource
- Enables optimization: `ANY_PATH` defines the path needed to access the resource. Thus, there is no need for Oracle XML DB to use an index to obtain a suitable path.

PATH_VIEW:

- Is created and defined in `catxdbpv.sql`
- Displays all the path names to a particular resource. For each unique path to access a resource, it contains a separate row.
- Displays the properties of the link (multiple paths of a resource)

Accessing Resources by Using SQL Access

```
DESCRIBE xdb.resource_view
```

```
SQL> DESCRIBE xdb.resource_view
Name          Null?    Type
-----        -----
RES           XMLTYPE(XMLSchema "http://xm
lns.oracle.com/xdb/XDBResour
ce.xsd" Element "Resource")
ANY PATH      VARCHAR2(4000)
RESID         RAW(16)
```

```
DESCRIBE xdb.path_view
```

```
SQL> desc xdb.path_view
Name          Null?    Type
-----        -----
PATH          VARCHAR2(1024 CHAR)
RES           XMLTYPE(XMLSchema "http://xm
lns.oracle.com/xdb/XDBResour
ce.xsd" Element "Resource")
LINK          XMLTYPE
RESID         RAW(16)
```



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

You can query both `resource_view` and `path_view` using the `DESCRIBE` command.

Accessing Resources by Using SQL Access: RESOURCE_VIEW and PATH_VIEW SQL Functions

UNDER_PATH SQL function

```
under_path(resource_column,  
          [depth],  
          pathname,  
          [correlation]);
```

EQUALS_PATH SQL function

```
equals_path(resource_column,  
            pathname);
```

PATH SQL function

```
path(correlation);
```

DEPTH SQL function

```
depth(correlation);
```



ORACLE

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

RESOURCE_VIEW and PATH_VIEW are used to access paths.

- **under_path SQL function:** Returns all the paths under the path that is specified in the first parameter. To speed up access when traversing down, it uses the hierarchical index of the Oracle XML DB Repository.
- **equals_path SQL function:** Finds the resource with the specified path name. It is functionally equivalent to `under_path` with a depth restriction of zero.
- **path SQL function:** Returns the relative path name corresponding to the absolute path of the resource that is specified in the path name argument
- **depth SQL function:** Returns the folder depth of the resource under the specified starting path

Parameters: `resource_column` is the column name or alias of the resource column and `pathname` is the name of the path to be resolved. `depth` is a positive integer that indicates the maximum depth to be searched. `correlation` can be used to correlate `under_path` with the related SQL functions (`path` and `depth`).

Determining Paths Under a Path by Using RESOURCE_VIEW: Absolute

Determining paths under a path: *absolute*

```
SELECT ANY_PATH
FROM RESOURCE_VIEW
WHERE under_path(RES, '/home/oe') = 1;
```

The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the results of the SQL query. The first section, labeled 'ANY_PATH', lists several absolute paths under the '/home/oe' directory. The second section, labeled '154 rows selected', lists paths that are not under the '/home/oe' directory.

ANY_PATH
/home/OE/PurchaseOrders
/home/OE/PurchaseOrders/2002
/home/OE/PurchaseOrders/2002/Apr
/home/OE/PurchaseOrders/2002/Apr/AMCEWEN-20021009123336171PDT.xml
/home/OE/PurchaseOrders/2002/Apr/AMCEWEN_20021009123336171PDT.xml

154 rows selected
/home/OE/purchaseOrder.xsl
/home/OE/xsd
/home/OE/xsl
/home/OE/xsl/empdept.xsl

ORACLE

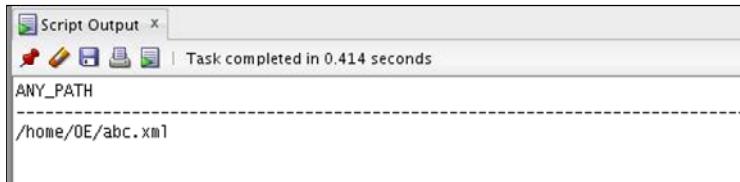
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

By using the ANY_PATH column, you can retrieve the *absolute* paths under the /home/oe path. If you use the unequal symbol (!=), you can find all the paths in the repository that are not under the /home/oe path.

Determining Paths Under a Path by Using RESOURCE_VIEW: Relative

Determining paths under a path: *relative*

```
SELECT ANY_PATH, path(21), path(22)
FROM RESOURCE_VIEW
WHERE under_path(RES, 1, '/home/xml', 21) = 1
OR under_path(RES, 1, '/home/oe', 22) = 1;
```



The screenshot shows the Oracle SQL Developer interface with a 'Script Output' window. The window title is 'Script Output X'. It displays the following output:
Task completed in 0.414 seconds
ANY_PATH

/home/0E/abc.xml

ORACLE

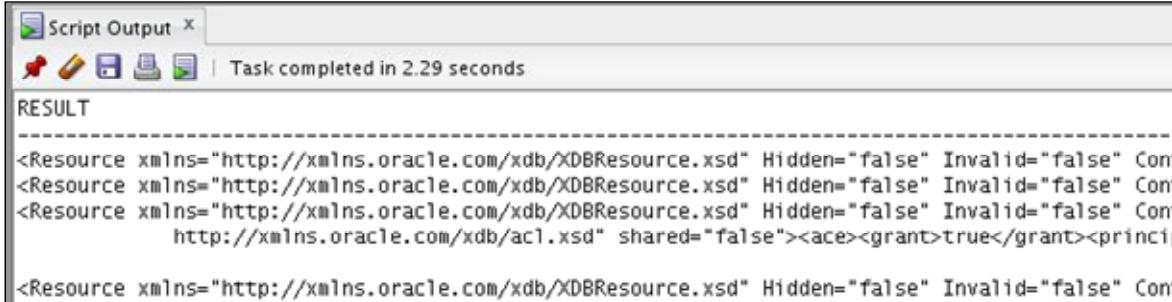
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Because the argument in the `path` function establishes a correlation with the third argument in the `under_path()` function in the query, the first and second path functions fetch the *relative* paths in accordance with the respective occurrences of the `under_path()` functions. The `under_path()` function returns all the paths under the `/home/xml` or `/home/oe` path.

Note: For additional information on accessing resources in Oracle XML DB Repository, see *Oracle XML DB Developer's Guide 12c Release 1 (12.1)*.

Extracting Resource Metadata by Using RESOURCE_VIEW

```
SELECT XMLQuery('declare namespace  
n1="http://xmlns.oracle.com/xdb/XDBResource.xsd";  
//n1:Resource' passing RES returning content)  
AS result  
FROM RESOURCE VIEW;
```

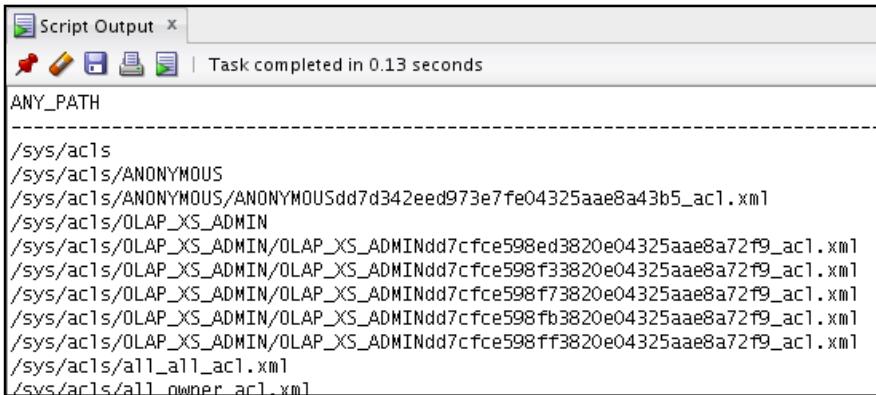


ORACLE

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

To retrieve the resource metadata, you can use the `XMLQUERY()` SQL function in conjunction with the `under path()` function.

```
SELECT ANY_PATH, XMLQUERY(' //Resource/CreationDate/text()' 
    passing RES returning content) AS creation_date
FROM RESOURCE_VIEW
WHERE under_path (RES, '/sys') = 1;
```



Extracting Link and Resource Information by Using PATH_VIEW

```

SELECT PATH,
       XMLQUERY('declare namespace
                  ns1="http://xmlns.oracle.com/xdb/XDBStandard";
                  /ns1:LINK/ns1:Name/text()' PASSING LINK RETURNING CONTENT)
          PATHNAME,
       XMLQUERY('declare namespace
                  ns1="http://xmlns.oracle.com/xdb/XDBStandard";
                  /ns1:LINK/ns1:ParentName/text()' PASSING LINK RETURNING
                  CONTENT) PARENTNAME,
       XMLQUERY('declare namespace
                  ns1="http://xmlns.oracle.com/xdb/XDBStandard";
                  /ns1:LINK/ns1:ChildName/text()' PASSING LINK RETURNING
                  CONTENT) CHILDNAME,
       XMLQUERY('declare namespace
                  ns1="http://xmlns.oracle.com/xdb/XDBResource.xsd";
                  /ns1:Resource/ns1:DisplayName/text()' passing RES returning
                  content) displayname
FROM PATH_VIEW
WHERE PATH LIKE '/sys/apps%';

```



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

PATH_VIEW is used to extract a link between a path and its parent. As shown in the slide, the extract function can be used to fetch various information about links.

When there are multiple links to the same resource, only those paths under the path name specified by the under_path() function are returned. Suppose that /a/b/c, /a/b/d, and /a/e are links to the same resource. A query on PATH_VIEW that retrieves all the paths in /a/b returns only /a/b/c and /a/b/d, and not /a/e.

The output of the code example in the slide is as follows (results can vary):

A screenshot of the Oracle SQL Developer interface showing a query result window. The window title is 'Query Result'. The toolbar icons include a green play button, a red error icon, a blue refresh icon, and a yellow warning icon. The status bar at the bottom left says 'SQL | All Rows Fetched: 1 in 0.626 seconds'. The result table has columns: PATH, PATHNAME, PARENTNAME, CHILDNAME, and DISPLAYNAME. There is one row of data: PATH is '/sys/apps', PATHNAME is 'apps', PARENTNAME is 'sys', CHILDNAME is 'apps', and DISPLAYNAME is 'apps'.

PATH	PATHNAME	PARENTNAME	CHILDNAME	DISPLAYNAME
1 /sys/apps	apps	sys	apps	apps

Lesson Agenda

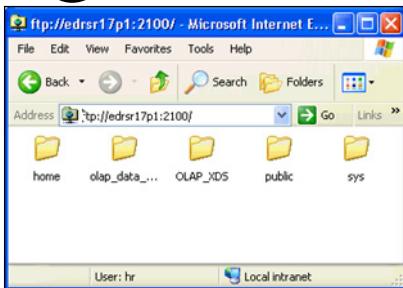
- XML DB Repository: overview
- Creating folders and resources using PL/SQL
- Accessing resources
 - SQL access
 - Navigational access
- Access control lists
- Compound documents
- Repository events



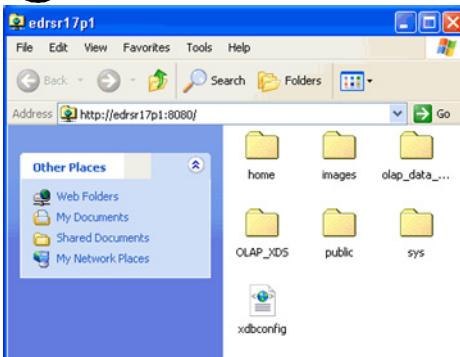
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Navigational Access

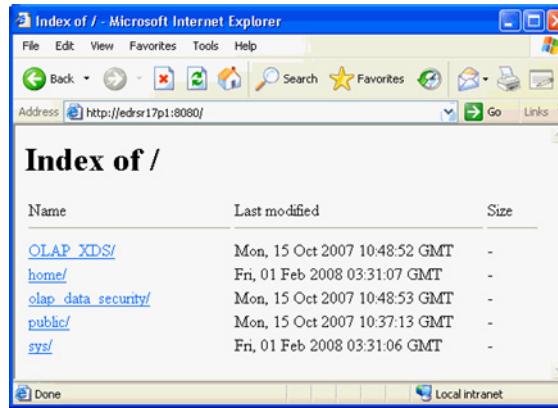
1 Using FTP



2 Using WebDAV



3 Using HTTP



ORACLE®

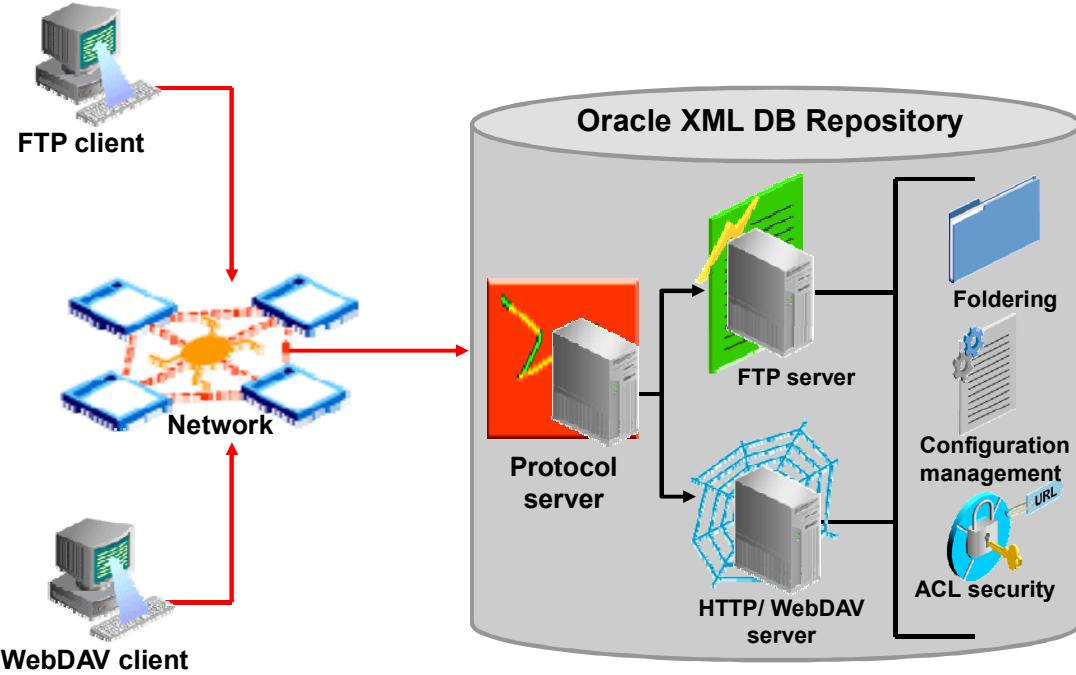
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle XML folders support the protocol standards that are used by many operating systems. As a result, an Oracle XML folder acts like a folder or directory in operating system environments. As shown in the slide, you can access the repository resources by using the following Internet protocols:

1. FTP
2. WebDAV
3. HTTP or HTTPS

These protocols enable XML data in the Oracle database to be accessed through commonly available desktop clients such as browsers, web folders, and MS Office. You can perform the required tasks (for example, editing content, creating directories, and navigating through different folders) with no additional software requirements.

Internet Access

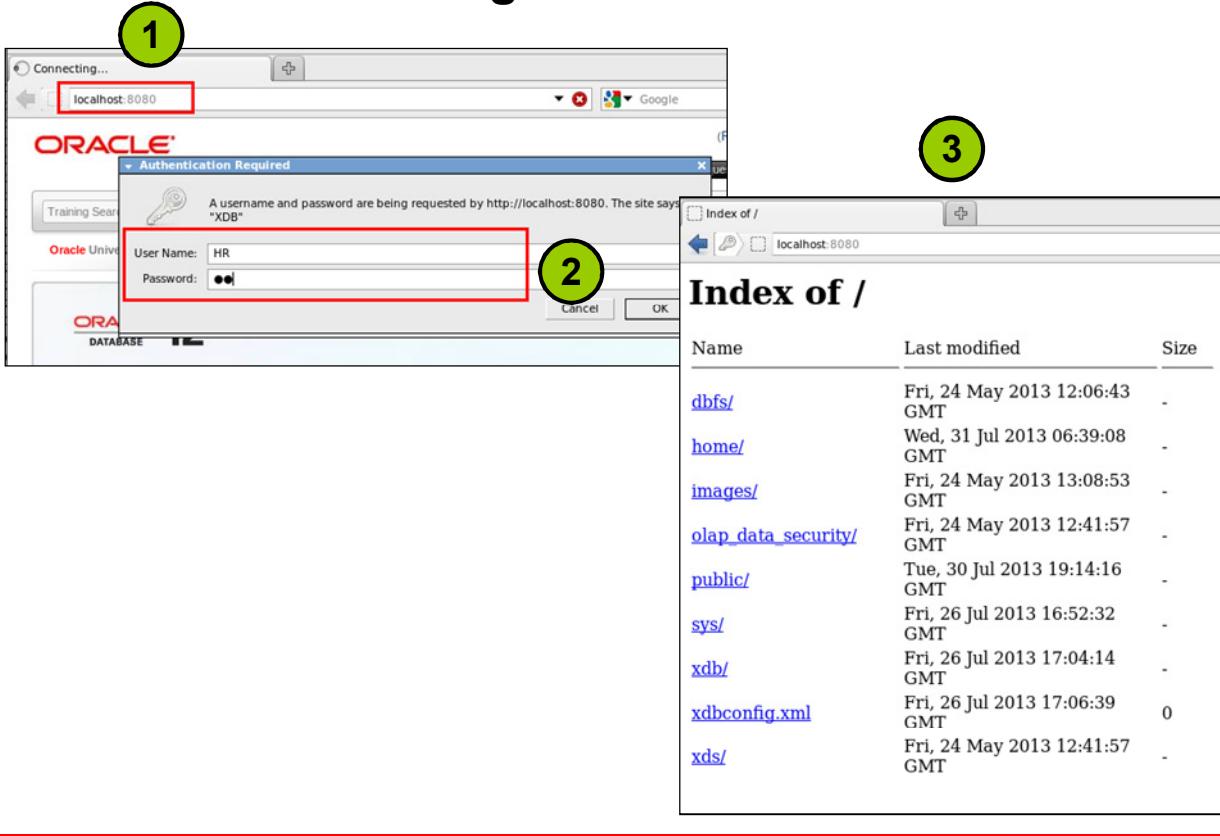


ORACLE®

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

The Oracle XML DB protocol server supports the FTP, WebDAV, and HTTPS protocols. When you insert an XML document into the database by using protocols such as FTP or WebDAV, the Oracle XML DB protocol server checks the validity of the inserted document against the XML schema specified by the document. If the incoming XML document is not based on the XML schema, it is stored as a binary document. The protocol servers are disabled by default.

Starting an HTTP Session



ORACLE

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

- To start an HTTP session by using your web browser such as Mozilla Firefox, enter either the machine name or the IP address followed by a colon, and the port number as a parameter to HTTP. The port number is usually 8080. In the slide example, we used localhost for the server.

Example

```
http://152.22.22.22:8080
```

- For the username and password, enter your Oracle Database username and password.

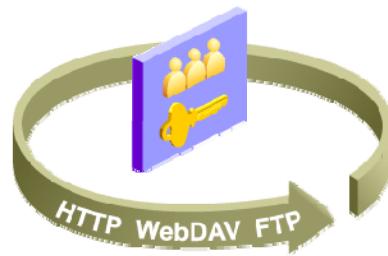
Note: In the slide example and on your machine, enter `HR` (uppercase) for the User Name, and `hr` (lowercase) for the password. That's how both are stored in the database.

- You are logged in to the Oracle XML DB Repository, where you can browse directories and download files in the same manner as you would in any GUI file manager.

Note: The ports for FTP and HTTP are disabled by default. You can modify them through the `xdbconfig.xml` configuration file or the EM Configuration link in XML Database.

HTTPS Support

- HTTPS is a secure HTTP protocol.
- Starting an HTTPS session is similar to starting an HTTP session.
- There are different port numbers for HTTP and HTTPS.



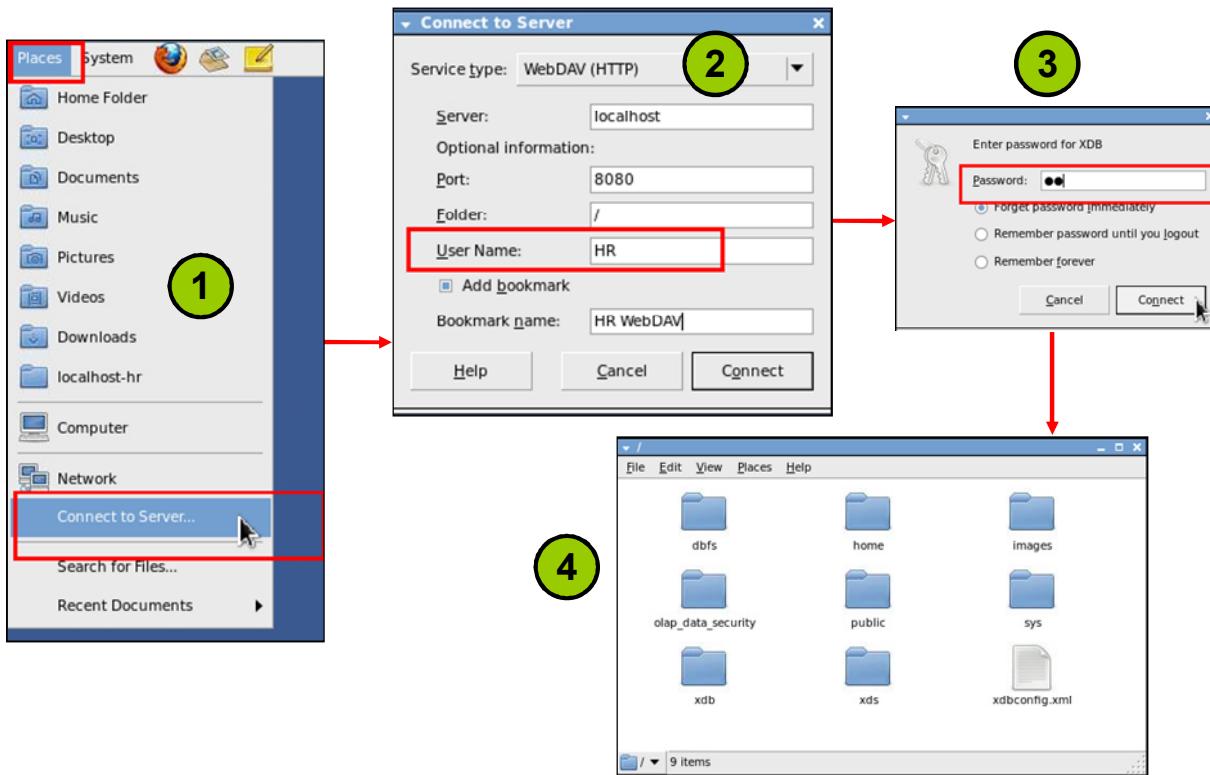
ORACLE

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

In releases before Oracle Database 10g, Release 2, Oracle XML DB supported only standard HTTP (TCP). In Oracle Database 10g, Release 2 and later, Oracle XML DB supports concurrent HTTP and HTTPS connections to the Oracle XML DB Repository. This enables the use of the secure HTTP protocol (HTTPS) in addition to the HTTP protocol, thereby adding support for industry-standard security standards to the Oracle XML DB protocol servers.

Oracle XML DB enables the HTTPS protocol to be used to secure the transmission of information between the client and the server.

Starting a WebDAV Session



ORACLE®

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

1. To create a web folder, from your desktop, select Places, and then select Connect to Server.
2. The Connect to Server window opens. Enter the following values and click Connect.
 - a. Service Type: WebDAV (HTTP)
 - b. Server: Your server name or localhost.
 - c. Port: Your HTTP_port_number
 - d. For the User Name, enter your Oracle Database User Name (case sensitive). In this example, the User Name is HR.
 - e. You can save a bookmark for this web folder. Click the Add bookmark check box, and then enter the Bookmark name. Next, click Connect.
3. You are prompted to enter your Password for User Name HR. Enter hr (in lowercase).
4. You are logged in to the Oracle XML DB Repository, where you can browse directories and download files in the same manner as you would in a Windows environment.

Quiz

An Oracle XML DB Repository resource contains only XML data.

- a. True
- b. False



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

Answer: b

Practice 14-1: Overview

This practice covers the following topics:

- Creating repository resources
- Managing and accessing repository resources



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

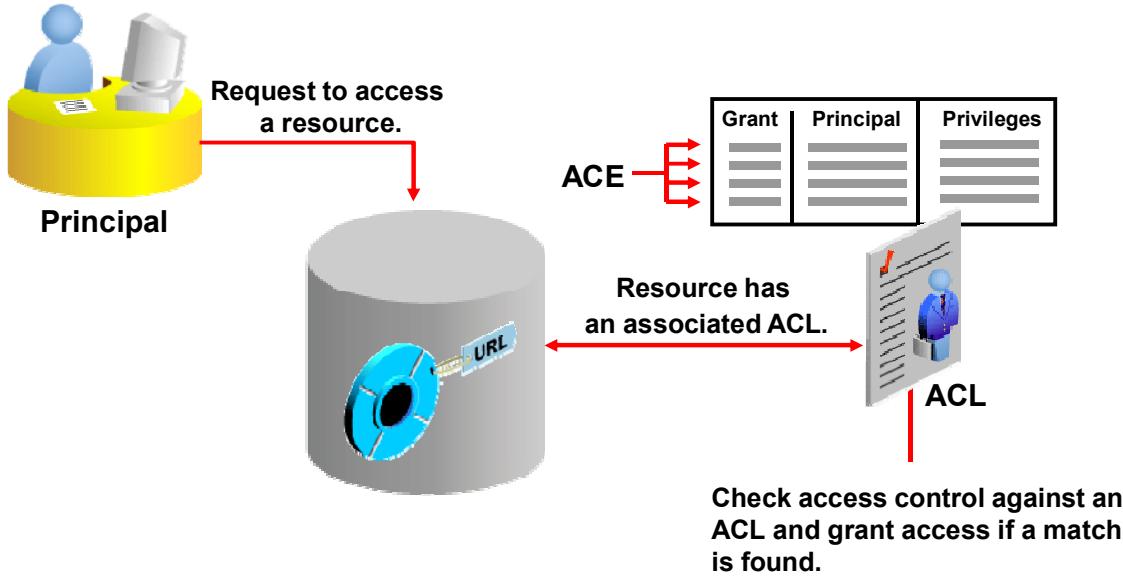
Lesson Agenda

- XML DB Repository: overview
- Creating folders and resources using PL/SQL
- Accessing resources
 - SQL access
 - Navigational access
- Access control lists
- Compound documents
- Repository events



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

Access Control Lists



ORACLE®

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

The access control list (ACL) is a security mechanism for the Oracle XML DB resources. It specifies a privilege-based access control for resources to users. Whenever a resource is accessed, a security check is performed. You can have only one ACL per resource. In Oracle XML DB, ACLs are stored in the `XMLType` table `XDB$ACL`. Every ACL conforms to the `acl.xsd` XML schema.

The ACL determines the privileges of a *principal*, which can be a database user or a group of database users who are granted a common role to access the methods for the resource. Each ACL has a list of access control entities (ACEs). An ACE has the following elements:

- A *Grant* (Boolean) that indicates whether or not this ACE grants or denies privileges
- A *principal* (either a user or a role) that indicates to whom the ACL applies
- A list of privileges that are granted or denied

The database collects the list of ACEs pertaining to the user who is logged in to the current database session. The list of currently active roles for the given user is maintained as a part of the session and is used to match ACEs, along with the current user's ID. To check access control against an ACL, you need the ID of the ACL and the owner of the object being secured. The Oracle XML DB hierarchy automatically associates an ACL ID and owner ID with an object that is mapped into its file system.

Any SQL application that wants ACL support can use the SQL API directly without using the Oracle XML DB Repository file system. But it must store the attributes that are necessary to use the API.

ACL-Based Security Management: Managing an ACL on a Resource

- To associate an ACL with a resource:

```
CALL DBMS_XDB_REPOSET.setACL('/home/xml/abc.xml',
    '/sys/acls/all_owner_acl.xml');
```

1

```
CALL DBMS_XDB.setACL('/home/xml/abc.xml', succeeded.
```

- To display the ACL that is associated with a resource:

```
SELECT DBMS_XDB_REPOSET.getACLDOMAIN('/home/xml/abc.xml')
AS result
FROM dual;
```

2

```
<acl description="Private:All privileges to OWNER only and not accessible to others" xmlns="http://xmlns.oracle.com/xdb/acl">
<ace>
<grant>true</grant>
<principal>dav:owner</principal>
<privilege>
<all/>
</privilege>
</ace>
</acl>
```

ORACLE

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

Functions and Procedures

- `setACL()`: This procedure sets the ACL on a resource. The example in the slide uses `all_owner_acl.xml`, which is a predefined ACL for restricting all the privileges on a resource (`abc.xml`) only to the `owner`.
- `getACLDOMAIN`: Retrieves the ACL document that protects a resource, given the path name of the resource. In the example, the `getACLDOMAIN` procedure shows that the `<principal>` user is `dav:owner`.

Output of getACLDocument

- Before setting ACL with setACL ():

RESULT

```
-----  
<acl description="Protected:Readable by PUBLIC and all  
privileges to OWNER" ... shared="false">  
...  
<ace>  
    <grant>true</grant>  
    <principal>PUBLIC</principal>  
    <privilege>  
        <read-properties/>  
        <read-contents/>  
        <read-acl/>  
        <resolve/>  
    </privilege>  
</ace>  
...  
</acl>
```

- After setting ACL with setACL ():

RESULT

```
-----  
<acl description="Private:All privileges to OWNER only and not  
accessible to others" ... shared="true">  
    <ace>  
        <grant>true</grant>  
        <principal>dav:owner</principal>  
        <privilege>  
            <all></all>  
        </privilege>  
    </ace>  
</acl>
```

ACL-Based Security Management: Managing Privileges

- To retrieve privileges associated with a resource:

```
SELECT
  DBMS_XDB_REPO.getPrivileges ('/home/xml/abc.xml/')
  AS result
FROM DUAL;
```

1

```
RESULT
-----
<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <read-properties/>
  <read-contents/>
  <write-config/>
```

- To change privileges associated with a resource:

```
CALL
  DBMS_XDB_REPO.changePrivileges ('/home/xml/abc.xml', a
  ce);
  . . .
```

2

ORACLE

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

The ACEs contain all the privileges for a particular user. There are two types of privileges that can be given to users:

- Aggregate:** Can contain a number of privileges. An aggregate is a naming convenience that simplifies usability when the number of privileges becomes large. These privileges control how users can operate on given resources.
- Atomic:** Cannot be subdivided

Functions and Procedures

- getPrivileges () : Returns all the privileges granted to the current user for a resource that can also be a folder.
- changePrivileges () : Adds an ACE to a resource ACL. Suppose that ace in the example is an XMLType with the following data:

```
<ace xmlns="http://xmlns.oracle.com/xdb/acl.xsd" . . .>
  <principal>OE</principal>
  <grant>true</grant>
  <privilege><all/></privilege>
</ace>'
```

Then, the ACL document for /home/xml/abc.xml before and after the call to changePrivileges is:

Before:

```
<acl description=". . .">
  <ace>
    <principal>dav:owner</principal>
    <grant>true</grant>
    <privilege><all/></privilege>
  </ace>
</acl>
```

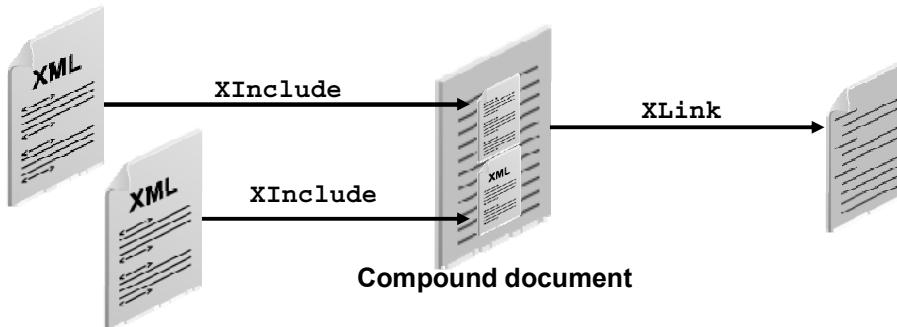
After:

```
<acl description=". . .">
  <ace>
    <grant>true</grant>
    <principal>dav:owner</principal>
    <privilege>
      <all></all>
    </privilege>
  </ace>
  <ace>
    <grant>true</grant>
    <principal>OE</principal>
    <privilege>
      <all></all>
    </privilege>
  </ace>
</acl>
```

Note: You can see the contents of the ACL document by using the getACLDocument code shown earlier in this lesson.

Compound Documents

- Compound documents are file resources that contain other documents or document fragments.
- XInclude and XLink are W3C recommendations that provide links to other documents.
 - XInclude: Merges multiple documents into a single document
 - XLink: Defines links between documents



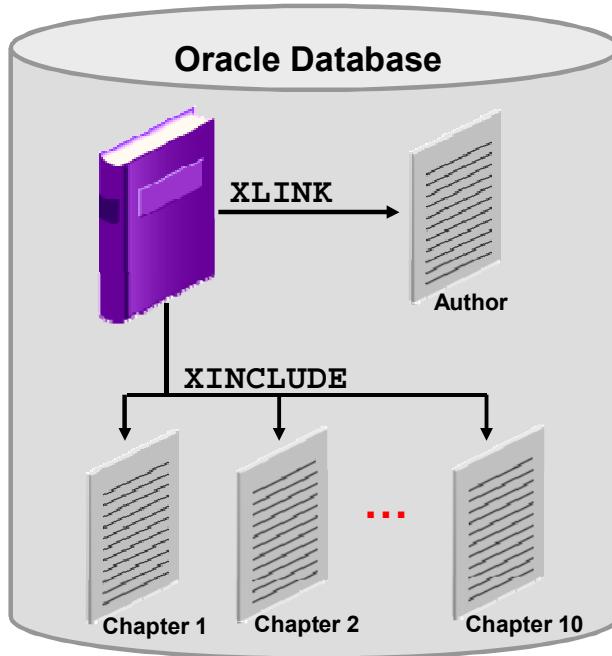
ORACLE

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

Compound documents are file resources that contain other documents or document fragments. XInclude and XLink are W3C recommendations that provide links to other documents.

- XInclude is used to merge the infosets of multiple XML documents into a single infoset. Its `xi:include` element is used to include another document by specifying its Uniform Resource Identifier (URI) as the value of an `href` attribute. The `xi:include` element can be nested; that is, an included document can itself include other documents. The `xi:include` element has another attribute, `xpointer`, that is used to include a fragment of any document. However, `xpointer` is not supported by Oracle XML DB.
- XLink is used to define a simple or extended link between documents. Simple links are unidirectional, from a source to a target. Extended links (sometimes called complex) define relationships between multiple documents, with different directionalities. For a simple XLink link (that is, the document containing the link), the source end must be an XML document, whereas the target end can be any document. However, there are no such restrictions for extended links. Oracle XML DB supports only simple links.

Compound Documents: Example



ORACLE®

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

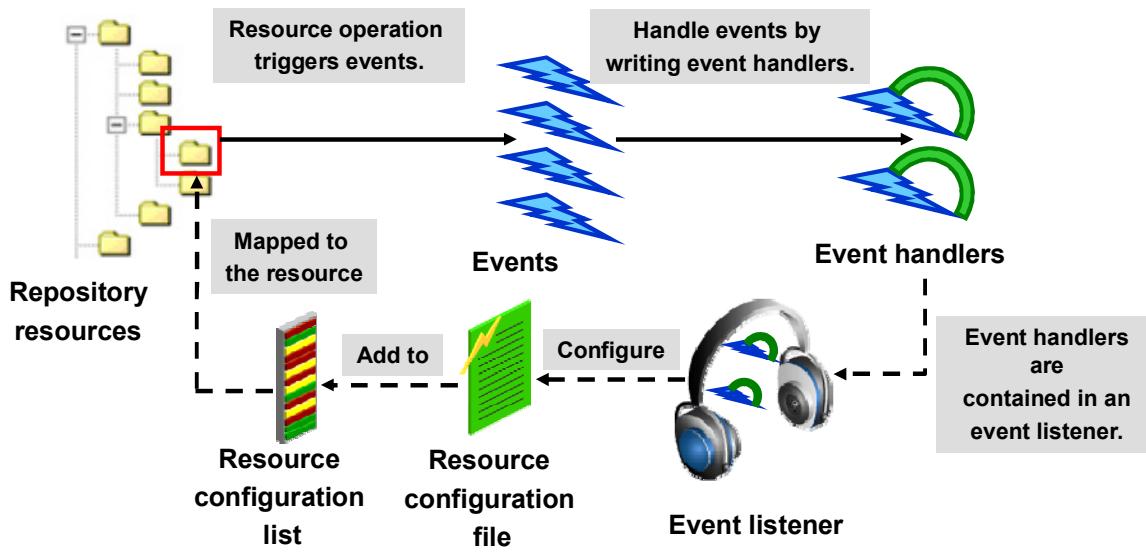
A book may be an example of a typical compound document. Each book has chapters, each of which can be managed as separate objects with their own URLs. A chapter document can have its own metadata and access control, and can be versioned. A book can include (reference) a specific version of a chapter document. The same chapter document can be included in multiple book documents for reuse. Because inclusion is modeled by using XInclude, content management is simplified. It is easy, for example, to replace one chapter in a book with another.

An author's information is better stored separately. Therefore, you can use XLINK to link the author information to your book document.

```
<Book xmlns:xi="http://www.w3.org/2001/Xinlcude"
      xmlns:xlink="http://www.w3.org/1999/xlink">
  <Author xlink:type="simple"
         xlink:href="/public/author/dir/paul.xml"> A. Paul</Author>
  <xi:include href="toc.xml"/>
  <xi:include href="chapter1.xml"/>
  <xi:include href="chapter2.xml"/>
  <xi:include href="index.xml"/>
</Book>
```

Repository Events

- Repository events are like triggers for the Oracle XML DB Repository operations.
- Event handlers can be written and configured to a resource or to the repository as a whole.



ORACLE

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

Repository events are like triggers for the Oracle XML DB Repository operations. They enable you to program your application to perform certain actions whenever a particular resource operation is executed. Repository resource operations include creating, deleting, locking, unlocking, rendering, linking, unlinking, placing under version control, checking in, checking out, unchecking out (reverting a checked out version), opening, and updating. Except for the render operation, each of these operations is associated with one or more *pairs* of events (pre and post).

You can write event handlers in PL/SQL or Java to handle an event that is “triggered.” Each event handler processes a single event. One or more event handlers are contained in an event listener. You can associate an event listener to the resource that you are concerned with, by mapping a resource configuration file to the resource. A resource can be configured by using multiple resource configuration files. These files are stored in a resource configuration list and they are processed in list order. Repository events can be associated to individual resources or to the entire repository.

Implementing Repository Events

Three things to do to implement a repository event:

1. Write an event handler.
2. Create a resource configuration file.
3. Append the resource configuration file to the resource configuration list.



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

To implement a repository event, you must perform the following:

1. You must write an event handler. You may do this in PL/SQL or Java.
2. You must create a resource configuration file.
3. You must append the resource configuration file that you created to the resource configuration list of the specified resource.

Note: For more details about configuring repository events, refer to the *Oracle XML DB Developer's Guide 12c, Release 1 (12.1)*.

Summary

In this lesson, you should have learned how to:

- Describe the key features and the architecture of the Oracle XML DB Repository
- Create folders and resources in the repository
- Use RESOURCE_VIEW and PATH_VIEW APIs to access the repository
- Use access control lists
- Describe compound documents
- Describe repository events



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

Practice 14-2: Overview

This practice covers the following topics:

- Working with an ACL
- Checking your understanding of the repository, compound documents, and repository events



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

You then use the RESOURCE_VIEW API and Internet protocols to view the resource's metadata and content. You also display the contents of ACLs, and view and change the ACLs attached to the repository resources.

There is also a quiz for you to check your understanding of the architecture of the Oracle XML DB Repository, the compound documents, and repository events.

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 Native Oracle XML DB Web Services

15

ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Identify what a web service is
- Configure web services for Oracle XML DB
- Enable web services for Oracle XML DB
- Query Oracle XML DB using a web service
- Access PL/SQL-stored procedures using a web service

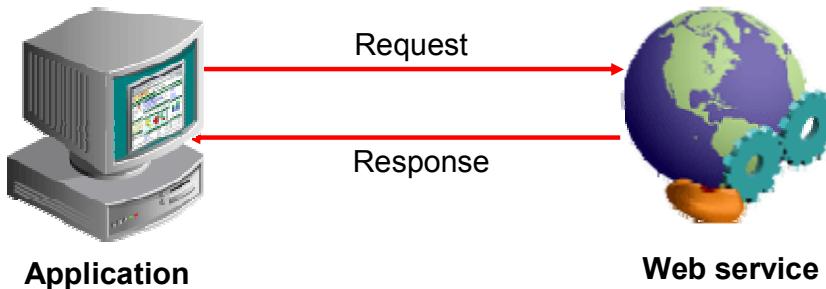


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

What Is a Web Service?

A web service:

- Is a software system or self-describing business function
- Is designed to support interoperable machine-to-machine interaction over a network
- Communicates with clients through standard protocols and technologies called web services



ORACLE

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

A service is essentially a process that performs a specific business function, such as validating a credit card submitted with a customer order. A suitable granular (modular) service design can enable a service to support many applications in an enterprise that spans platforms and organizational components. For example, to find the number of units sold for a particular product, the business function might need to query a sales management, a sales order, and a product inventory system within an organization.

A web service is a service that performs discrete business functions. Interaction with a web service is based on a set of standard protocols and technologies called web services. Web service:

- Refers to a standard set of platform-independent messaging protocols (SOAP, HTTP, and JMS)
- Allows connections between services from any Web-connected device
- Exchanges data and functionality in XML format

Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents. Web services have become an industry standard for exchanging information and giving access to business logic.

Web Service Standards

The most common web service standards are:

- SOAP:
 - Describes the message format of a web communication
 - Is platform and language independent
 - Is based on XML
- Web Services Description Language (WSDL):
 - Defines mechanisms for describing Web service operations in a platform-neutral way
 - Is based on XML
- Universal Description, Discovery, and Integration (UDDI)
 - Facilitates the registration and organization of Web service descriptions into a searchable hierarchy
 - Communicates via SOAP



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

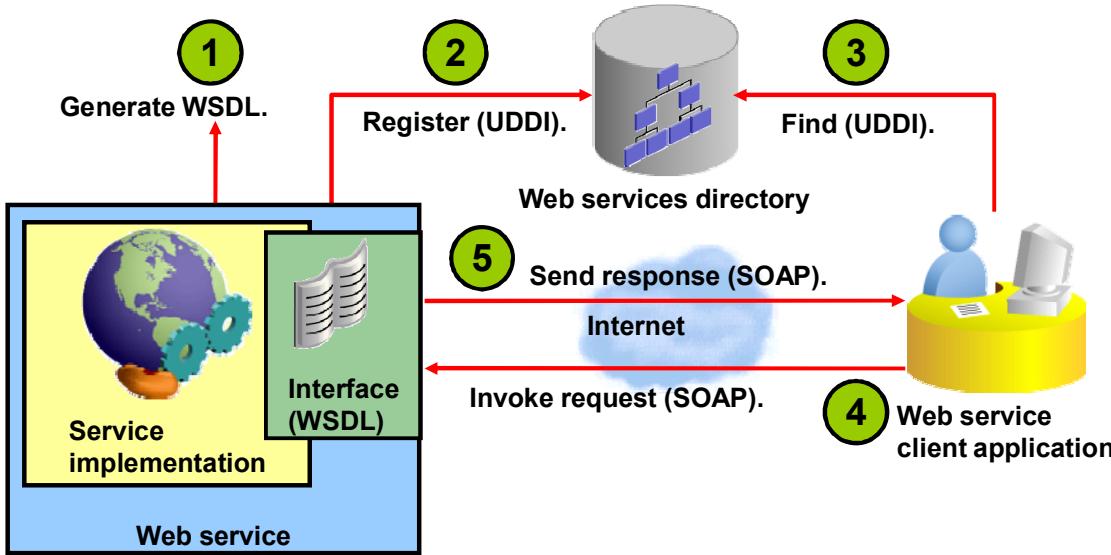
The standards for web services include a SOAP for message format, WSDL for service description, and UDDI for the registration and location of web services.

SOAP is an XML-based message format that is an open standard defined by the W3C. It is a mechanism for invoking operations and exchanging documents between applications as an envelope protocol that is embedded in other standard protocols.

The WSDL also uses an XML-based format, but it is used for describing the interface of a web service. This document includes a description of the endpoint, location, protocol binding, operations, parameters, and data types of all aspects of a web service.

UDDI is a set of specifications that defines the format of a standard registry for web services, as well as the protocol for messages used to access the registry. It also identifies the classifications and taxonomies for organizing web services.

Web Service Architecture



ORACLE®

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

The diagram in the slide shows a conceptual architecture of web services. The **provider** of a web service performs the following steps to expose a web service endpoint:

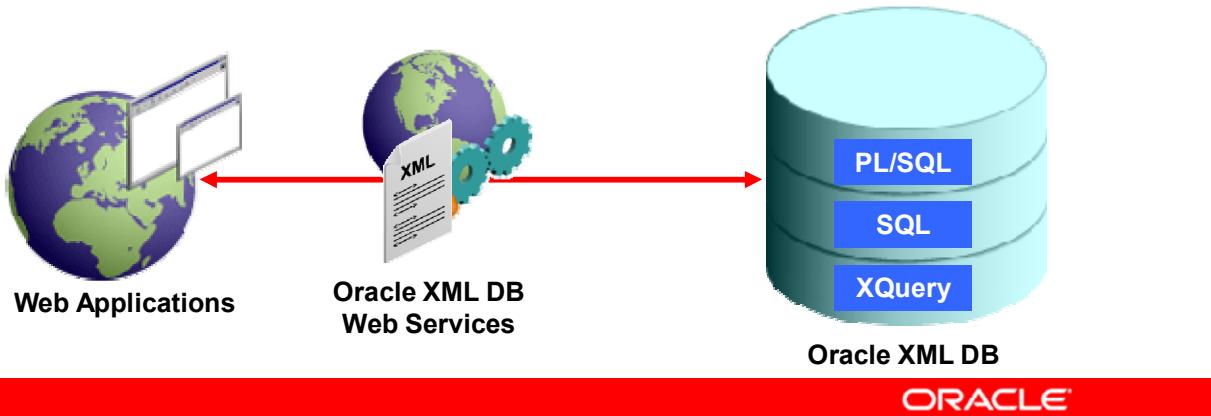
1. **Generates the WSDL:** The web service is developed and the XML format describing the service (the WSDL) is generated.
2. **Registers with the Web service registry:** The web service provider registers itself as a business entity and publishes the WSDL in one or more UDDI registries (for example, Oracle Service Registry).

The consumer of a web service performs the following steps to invoke a web service:

3. Searches the **UDDI registry** to locate the WSDL for the Web service
4. Invokes the **Web service** by sending a SOAP request by using the information stored in the WSDL. The WSDL information, which is obtained from the UDDI registry or WSIL browser, includes:
 - The URL for the service endpoint to locate the service functions
 - The WSDL that defines the interface methods and messages provided by the service
5. **Communicates the response** with a SOAP message that communicates information to the consumer. The SOAP request and response are typically transmitted using the simple HTTP request/response protocol.

Oracle XML DB Web Service: Overview

- Oracle Database supports native implementation of web services in the database.
- You write and deploy web services that can query the database by using SQL or XQuery, or access the stored PL/SQL procedures and functions.
- SOAP 1.1 is the version supported by Oracle XML DB.



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

ORACLE

Using the native Oracle XML DB Web services capability, your web applications access the database. You write and deploy web services that query the database by using SQL or XQuery, or access the stored PL/SQL functions and procedures. The logic for web services deployment resides in Oracle XML DB.

Starting in Oracle Database 11g, you use web service to perform the following:

- Query Oracle XML DB
- Access the PL/SQL-stored functions and procedures

The main protocol defining the kind of communication to a web service is Simple Object Access Protocol (SOAP). **Oracle XML DB supports SOAP 1.1 version.**

Why Native Oracle XML DB Web Services?

- No coding effort is required.
- Native database web services leverage the Oracle XML DB HTTP server.
- Native database web services leverage the powerful XML capabilities of Oracle XML DB to perform the following:
 - Automatically generate the WSDL.
 - Parse and process the SOAP request.
 - Generate the SOAP response.



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

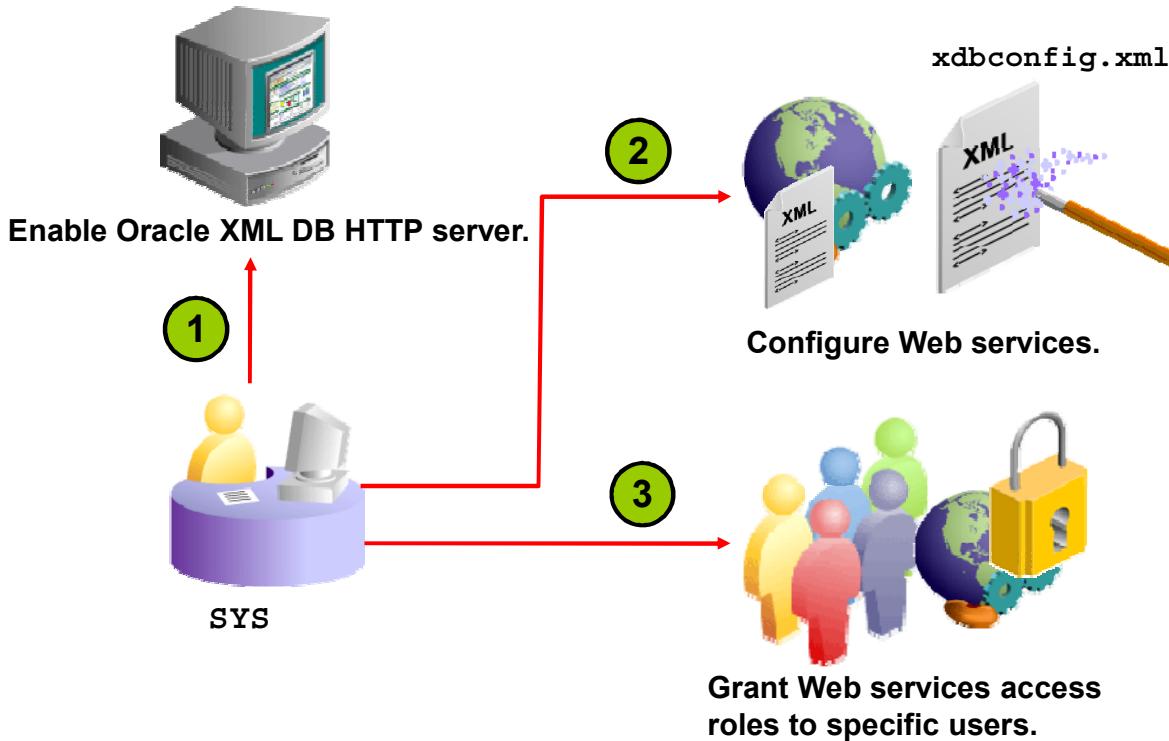
Native database web services allow Oracle Database 11g to directly participate in a service-oriented architecture. In the previous releases, creating a web service required hand coding or generating programs necessary to expose a SOAP endpoint, and then deploying this code by using an application server. These programs needed to address all aspects of a web service's usage such as the following:

- Generating the WSDL document
- Parsing and processing the SOAP request
- Arranging the required data into the correct XML format to provide the SOAP response

In Oracle Database 11g, native database web services eliminate the complexity for web services that provide access to data stored in an Oracle database. Native database web services allow you to expose PL/SQL-stored procedures, functions, and packages as web services with zero coding and zero deployment effort.

The database HTTP server, provided as part of Oracle XML DB Repository, allows these web services to be accessed using HTTP and HTTPS, without any additional application server infrastructure.

Steps Required to Use Web Services with Oracle XML DB



ORACLE

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

Configuring and Enabling Web Services for Oracle XML DB

For security reasons, Oracle XML DB is not preconfigured with the native Web services enabled. To make native Oracle XML DB web services available, you must have the Oracle XML DB HTTP server up and running, and you must explicitly add Web service configuration. Then, to allow specific users to use web services, you must grant them appropriate roles.

Configuring Oracle XML DB Using `xdbconfig.xml`

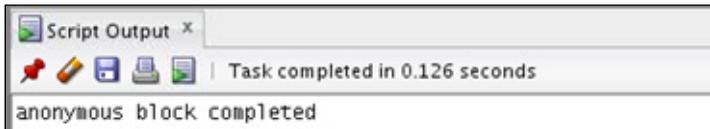
Oracle XML DB is managed internally through a configuration file, `/xdbconfig.xml`, which is stored as a resource in Oracle XML DB Repository. As an alternative to using Oracle Enterprise Manager to configure Oracle XML DB, you can configure it directly by using the Oracle XML DB configuration file.

The configuration file can be modified at run time. Updating the configuration file creates a new version of this repository resource. At the start of each session, the current version of the configuration file is bound to that session. The session uses this configuration file version for its duration, unless you make an explicit call to refresh the session to the latest version.

Note: For additional information about the configuration file, see chapter 34 titled “Administering Oracle XML DB,” in the *Oracle XML DB Developer’s Guide 11g Release 2 (11.2)* reference.

Adding a Web Services Configuration Servlet

```
DECLARE
  SERVLET_NAME VARCHAR2(32) := 'orawsrv';
BEGIN
  DBMS_XDB_REPO.deleteServletMapping(SERVLET_NAME);
  DBMS_XDB_REPO.deleteServlet(SERVLET_NAME);
  DBMS_XDB_REPO.addServlet(NAME => SERVLET_NAME, LANGUAGE => 'C',
    DISPNAME => 'Oracle Query Web Service',
    DESCRIPT => 'Servlet for issuing queries as a Web Service',
    SCHEMA => 'XDB');
  DBMS_XDB_REPO.addServletSecRole(SERVNAME => SERVLET_NAME,
    ROLENAME => 'XDB_WEBSERVICES', ROLELINK =>
    'XDB_WEBSERVICES');
  DBMS_XDB_REPO.addServletMapping(PATTERN => '/orawsrv/*',
    NAME => SERVLET_NAME);
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.

The code example in the slide shows you how to use procedures in the DBMS_XDB_REPO package to add the web services configuration servlet. You can also configure security settings for the web services by using the xdbconfig.xml Oracle XML DB configuration file. **You must be logged in as the sys user to perform these tasks.**

Verifying the Addition of a Web Services Configuration Servlet

```
SET LONG 10000
SQL> XQUERY declare default element namespace
"http://xmlns.oracle.com/xdb/xdbconfig.xsd";
for $doc in
fn:doc("/xdbconfig.xml")/xdbconfig/sysconfig/protocolconfig/ht
tpconfig/webappconfig/servletconfig/servlet-
list/servlet[servlet-name='orawsrv']
return $doc
/
Result Sequence
-----
<servlet xmlns="http://xmlns.oracle.com/xdb/xdbconfig.xsd">
  <servlet-name>orawsrv</servlet-name>
  <servlet-language></servlet-language>
  <display-name>Oracle Query Web Service</display-name>
  <description>Servlet for issuing queries as a Web Service</description>
  <servlet-schema>XDB</servlet-schema>
  <security-role-ref>
    <description/>
    <role-name>XDB_WEBSERVICES</role-name>
    <role-link>XDB_WEBSERVICES</role-link>
  </security-role-ref>
Result Sequence
-----
</servlet>
```



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

To verify the successful configuration of the Web service, execute the XQUERY command from SQL*Plus using a terminal window.

Note: The code example in the slide is entered as a single long line.

Granting Access to Web Services

- Grant basic access.

```
GRANT XDB_WEBSERVICES TO OE;
```

- Enable use of Web services over HTTP (not just HTTPS).

```
GRANT XDB_WEBSERVICES_OVER_HTTP TO OE;
```

- Enable access to PUBLIC objects:

```
GRANT XDB_WEBSERVICES_WITH_PUBLIC TO OE;
```



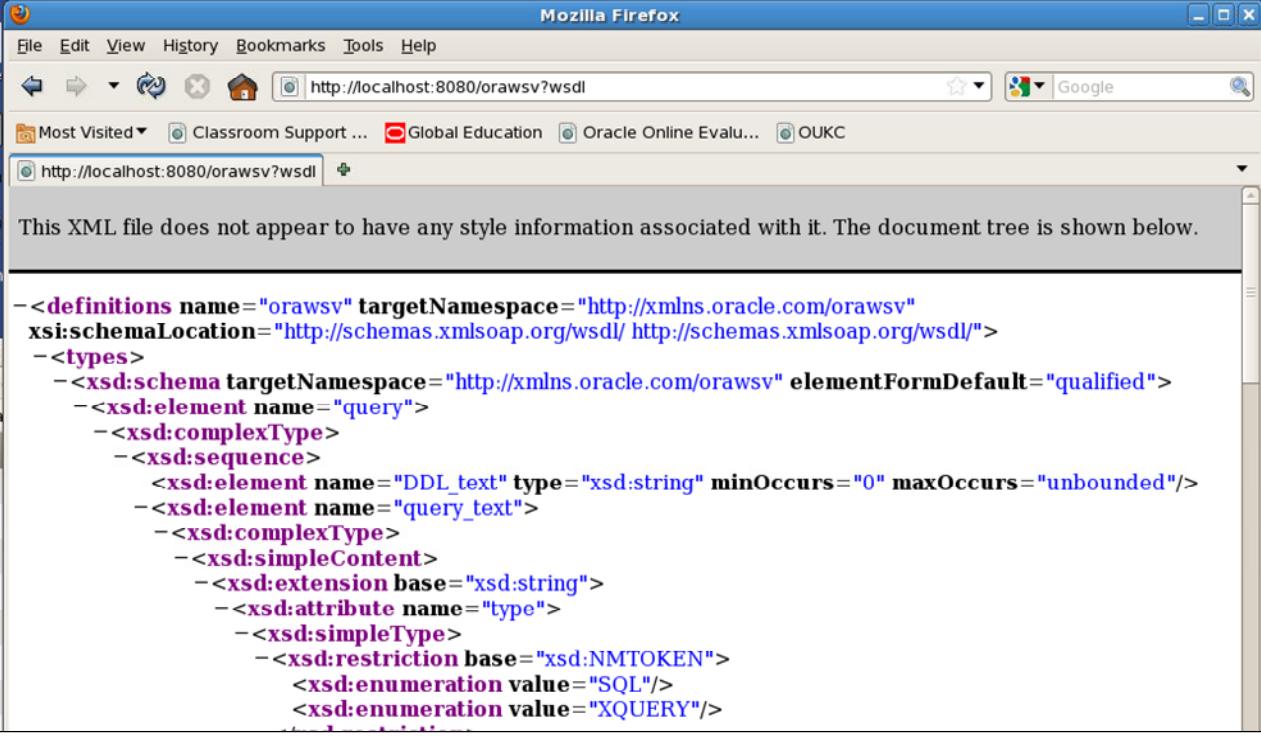
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Enabling Web Services for Oracle XML DB

After you have created a web service as the `SYS` user, to enable web services for a specific user, log on as the `SYS` user and grant the `XDB_WEBSERVICES` role to the user. To enable use of web services over HTTP (not just HTTPS), grant `XDB_WEBSERVICES_OVER_HTTP`.

By default, web services cannot access objects with the privileges granted to `PUBLIC`. To enable access to such objects, grant `XDB_WEBSERVICES_WITH_PUBLIC` to the user.

Viewing the WSDL Using HTTP



The screenshot shows a Mozilla Firefox browser window. The address bar contains the URL <http://localhost:8080/oraws?wsdl>. The page content displays the XML structure of the WSDL document. The XML code is as follows:

```
<definitions name="oraws" targetNamespace="http://xmlns.oracle.com/oraws" xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/ http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema targetNamespace="http://xmlns.oracle.com/oraws" elementFormDefault="qualified">
      <xsd:element name="query">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="DDL_text" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="query_text">
              <xsd:complexType>
                <xsd:simpleContent>
                  <xsd:extension base="xsd:string">
                    <xsd:attribute name="type">
                      <xsd:simpleType>
                        <xsd:restriction base="xsd:NMTOKEN">
                          <xsd:enumeration value="SQL"/>
                          <xsd:enumeration value="XQUERY"/>
                        </xsd:restriction>
                      </xsd:simpleType>
                    </xsd:attribute>
                  </xsd:extension>
                </xsd:simpleContent>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </types>
</definitions>
```

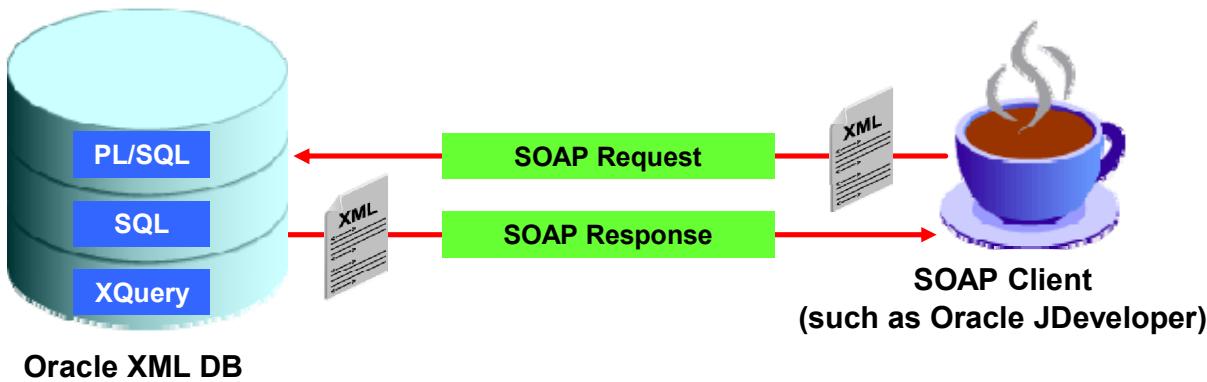


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The database queries that your web application sends to the web service are in the form of XML documents. These XML documents should conform to the XML schema listed in the WSDL.

Each web service has a WSDL document associated with it that specifies the formats of the incoming and outgoing XML documents using XML schema. This WSDL document is located at the URL <http://host:port/oraws?wsdl>, where the host and the port are the host and HTTPS port properties of your database. Enter oe as the username and oe as the password.

Using Native Oracle XML DB Web Services



ORACLE

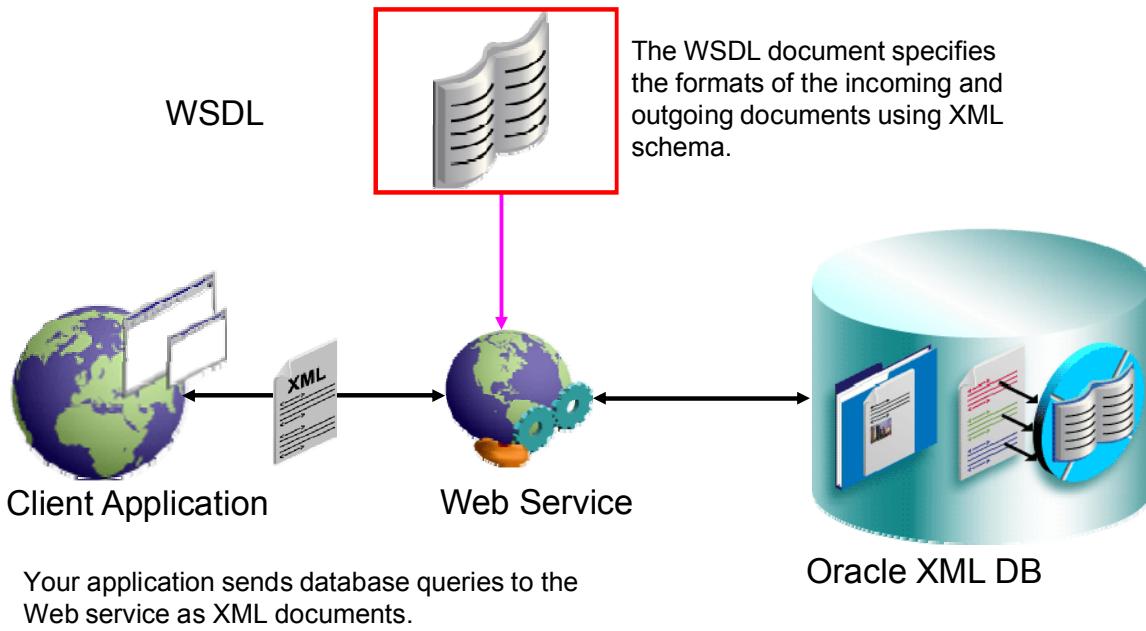
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The graphic in the slide shows a complete round trip of a web service call to the database. You can customize the input and output document formats when you use the web service that provides access to all PL/SQL-stored functions and procedures. The WSDL is automatically generated by the native database web services engine.

Note

- The WSDL is an XML language for describing the syntax of web service interfaces and their locations. The WSDL specification calls it “an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.”
- HTTP Analyzer is a SOAP client. You can also generate soap classes using Oracle JDeveloper.

Querying Oracle XML DB By Using a Web Service



ORACLE®

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

The Oracle XML DB web service for database queries:

- Is located at <http://host:port/oraws>
- Has a WSDL document that specifies the formats of the incoming and outgoing documents using XML schema

The WSDL document is located at <http://host:port/oraws?wsdl>.

Note: Host and port are the host and HTTPS port properties of your database. Your application sends database queries to the web service as XML documents that conform to an XML schema. This XML schema is contained in the WSDL document.

The following are some of the important parts of the incoming query documents:

- **query_text:** The text of your query. The type attribute specifies the type of your query that is either SQL or XQuery.
- **bind:** A scalar bind variable value. The name attribute names the variable.
- **bindXML:** An XMLType bind-variable value

- **max-rows**: The maximum number of rows to output for the query. By default, all rows are returned.
- **pretty_print**: Whether the output document should be formatted for pretty-printing. The default value is `true`, which means that the document will be pretty-printed. When the value is `false`, no pretty-printing is done, and the output rows are not broken with newline characters.

Example: You connect to `http://host:port/oraws`, and post the following XML document:

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/06/soap-envelope">
<env:Body>
<query xmlns="http://xmlns.oracle.com/oraws">
<query_text type="SQL">
<![CDATA[SELECT * FROM employees WHERE salary = :e]]>
</query_text>
<bind name="e">8300</bind>
<pretty_print>false</pretty_print>
</query></env:Body>
</env:Envelope>
```

Observe that in the input XML document, the query text is enclosed in `<![CDATA [...]]>`. This is appropriate to escape characters such as “`<`” and “`>`”. The `bind` element is used to bind the value `8300` to the bind variable named `e`. The `pretty_print` element turns off pretty-printing of the output.

The following document is the output XML document:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2002/06/soap-envelope">
<soap:Body>
<ROWSET><ROW><EMPLOYEE_ID>206</EMPLOYEE_ID><FIRST_NAME>William</FIRST_NAME><LAST_NAME>G
ietz</LAST_NAME><EMAIL>WGIETZ</EMAIL><PHONE_NUMBER>515.123.8181</PHONE_NUMBER><HIRE_DATE>07-JUN-
94</HIRE_DATE><JOB_ID>AC_ACCOUNT</JOB_ID><SALARY>8300</SALARY><MANAGER_ID>205</MANAGER_ID
><DEPARTMENT_ID>110</DEPARTMENT_ID></ROW></ROWSET>
</soap:Body>
</soap:Envelope>
```

Additional reference: See the *Using Oracle XML DB Web Services for Service-Oriented Architecture OBE* at:

[\[http://www.oracle.com/technology/obe/11gr1_db/datamgmt/xmldb2_b/xmldb2_b.htm#ws\]](http://www.oracle.com/technology/obe/11gr1_db/datamgmt/xmldb2_b/xmldb2_b.htm#ws)

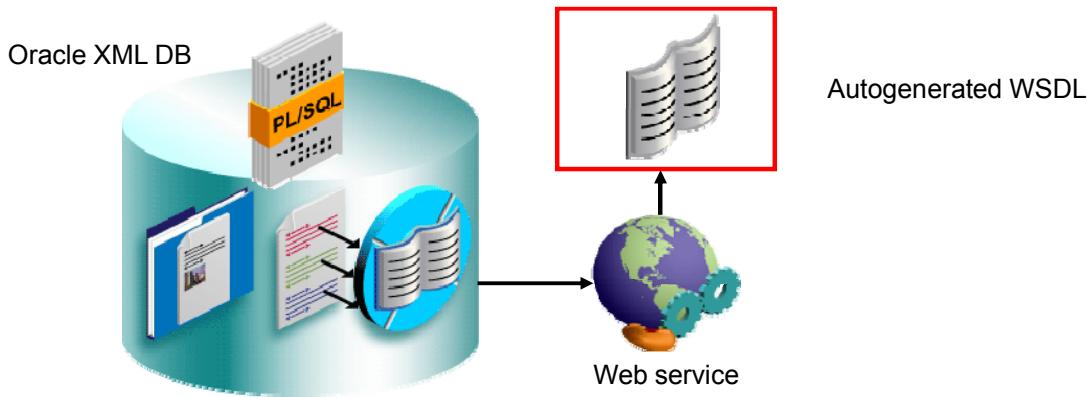
The incoming query document should conform to the following XML schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xdb="http://xmlns.oracle.com/xdb"
        targetNamespace="http://xmlns.oracle.com/orawsrv">
<element name="query">
<complexType><sequence>
<element name="query_text">
<complexType>
<simpleContent>
<extension base="string"> <attribute name="type"> <simpleType>
<restriction base="NMTOKEN">
<enumeration value="SQL"/> <enumeration value="XQUERY"/>
</restriction>
</simpleType>
</attribute>
</extension>
</simpleContent>
</complexType>
</element>
<choice maxOccurs="unbounded">
<element name="bind">
<complexType>
<simpleContent>
<extension base="string">
<attribute name="name" type="string"/>
</extension>
</simpleContent>
</complexType>
</element>
<element name="bindXML" type="any"/>
</choice>
<element name="null_handling" minOccurs="0">
<simpleType>
<restriction base="NMTOKEN">
<enumeration value="DROP_NULLS"/>
<enumeration value="NULL_ATTR"/>
<enumeration value="EMPTY_TAG"/>
</restriction>
</simpleType>
</element>
<element name="max_rows" type="positiveInteger" minOccurs="0"/>
<element name="skip_rows" type="positiveInteger" minOccurs="0"/>
<element name="pretty_print" type="boolean" minOccurs="0"/>
<element name="indentation_width" type="positiveInteger"
minOccurs="0"/>
<element name="rowset_tag" type="string" minOccurs="0"/>
```

```
<element name="row_tag" type="string" minOccurs="0"/>
<element name="item_tags_for_coll" type="boolean" minOccurs="0"/>
</sequence>
</complexType> </element> </schema>
```

Accessing PL/SQL-Stored Procedures Using a Web Service

- You can use web services to access PL/SQL-stored procedures and functions.
- Each PL/SQL-stored function or procedure is associated with a separate, dynamic web service that has its own, generated WSDL document.



ORACLE

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

The Oracle XML DB web service for accessing PL/SQL-stored functions and procedures is located at http://host:port/orawserv/dbschema/package/fn_or_proc. The Oracle XML DB Web service for accessing PL/SQL function or procedure that is not in a package (stand-alone) is at http://host:port/orawserv/dbschema/fn_or_proc.

In the URL:

- “host” and “port” are the host and HTTPS port properties of your database
- “fn_or_proc” is the stored function or procedure name
- “package” is the package name
- “dbschema” is the database schema owning that package

Your input XML document must contain the inputs needed by the function or procedure. The output XML document contains the values of all `OUT` variables, as well as the return value.

The names of the XML elements in the input and output documents correspond to the variable names of the function or procedure. You see the exact XML element names in the generated WSDL document.

Defining a PL/SQL Function for Accessing the Web Service

```
CREATE OR REPLACE PACKAGE SALARY_CALCULATOR AUTHID CURRENT_USER AS
  FUNCTION TOTALDEPARTMENTSALARY (dept_id IN NUMBER DEFAULT 20,
  dept_name IN OUT VARCHAR2, nummembers OUT NUMBER)
  RETURN NUMBER;
END salary_calculator;

CREATE OR REPLACE PACKAGE BODY SALARY_CALCULATOR AS
  FUNCTION TOTALDEPARTMENTSALARY (dept_id IN NUMBER DEFAULT 20,
  dept_name IN OUT VARCHAR2, nummembers OUT NUMBER)
  RETURN NUMBER IS
    sum_sal NUMBER;
    BEGIN
      SELECT SUM(salary) INTO sum_sal FROM employees
      WHERE department_id = dept_id;
      SELECT department_name INTO dept_name FROM departments
      WHERE department_name = dept_name;
      SELECT count(*) INTO nummembers FROM employees
      WHERE department_id = dept_id;
      RETURN sum_sal;
    END;
  END;
```

```
PACKAGE SALARY_CALCULATOR compiled
PACKAGE BODY SALARY_CALCULATOR compiled
```



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

Now that you have a web service configured, create a PL/SQL program unit that the web service will call.

Viewing the PL/SQL Function's WSDL

The screenshot shows a Mozilla Firefox window displaying an XML document. The URL in the address bar is `http://127.0.0.1:8080/orawsrv/HR/SALARY_CALCULATOR/TOTALDEPARTMENTSALARY?wsdl`. The page content is an XML WSDL document. A red box highlights a specific section of the XML code:

```
<definitions name="TOTALDEPARTMENTSALARY" targetNamespace="http://xmlns.oracle.com/orawsrv/HR/SALARY_CALCULATOR/TOTALDEPARTMENTSALARY">
  <types>
    <xsd:schema targetNamespace="http://xmlns.oracle.com/orawsrv/HR/SALARY_CALCULATOR/TOTALDEPARTMENTSALARY" elementFormDefault="qualified">
      <xsd:element name="SNUMBER-TOTALDEPARTMENTSALARYInput">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="NUMMEMBERS-NUMBER-OUT">
              <xsd:complexType/>
            </xsd:element>
            <xsd:element name="DEPT_NAME-VARCHAR2-INOUT" type="xsd:string"/>
            <xsd:element name="DEPT_ID-NUMBER-IN" minOccurs="0" maxOccurs="1" type="xsd:double"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="TOTALDEPARTMENTSALARYOutput">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="RETURN" type="xsd:double"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </types>
</definitions>
```

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

After you create a function, you can view its corresponding WSDL that Oracle XML DB automatically generates, at the following URL:

`http://host:port/orawsrv/HR/SALARY_CALCULATOR/TOTALDEPARTMENTSALARY?wsdl`.

If `TotalDepartmentSalary` is a stand-alone function, the URL would have been
`http://host:port/orawsrv/HR/TOTALDEPARTMENTSALARY`.

The input XML document must contain the inputs needed by the PL/SQL program unit. Similarly, the output XML document contains the values of all OUT variables, as well as the return value. The names of the XML elements in the input and output documents correspond to the variable names of the function or procedure. The generated WSDL document shows you the exact XML element names.

Quiz

Which of the following defines the mechanisms for describing the web service operations in a platform-neutral way?

- a. SOAP
- b. WSDL
- c. UDDI



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

Answer: b

Quiz

Each PL/SQL-stored function or procedure is associated with a separate, dynamic web service that has its own, generated WSDL document.

- 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:

- Identify what a web service is
- Configure web services for Oracle XML DB
- Enable web services for Oracle XML DB
- Query Oracle XML DB using a web service
- Access PL/SQL-stored procedures using a web service



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

Practice 15 Overview: Creating Web Services for Oracle XML DB

This practice covers the following topics:

- Configuring web services for Oracle XML DB
- Granting access
- Define a PL/SQL function for accessing the web service and view it's WSDL
- Query Oracle XML DB using a web service



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

16

Exporting and Importing XML Data

ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Load XMLType data by using SQL*Loader
- Export and import XMLType data by using Oracle Data Pump



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

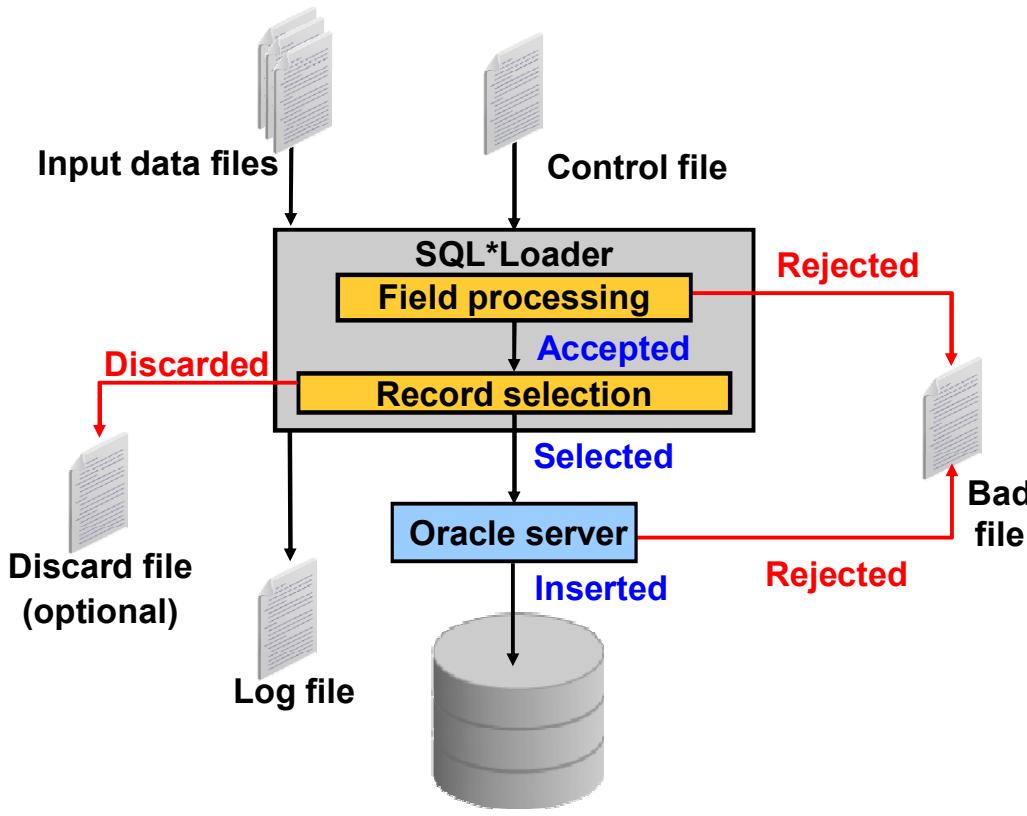
Lesson Agenda

- SQL*Loader
 - Overview
 - Loading XMLType data
- Oracle Data Pump
 - Overview
 - Exporting and importing XMLType data



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

SQL*Loader: Overview



ORACLE

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

SQL*Loader is a utility provided by Oracle database. It is the primary method for quickly populating Oracle Database tables with data from external files. It has a powerful data parsing engine that puts little limitation on the format of the data in the data file. SQL*Loader can be used to load data across a network, from disk, from tape, or from a named pipe.

The files that are used by SQL*Loader are as follows:

Input data files: SQL*Loader reads data from one or more files (or operating system equivalents of files) that are specified in the control file. From SQL*Loader's perspective, the data in the data file is organized as records. A particular data file can be in a fixed record format, variable record format, or stream record format. The record format can be specified in the control file with the `INFILE` parameter. If no record format is specified, the default stream record format is used.

Control file: The control file is a text file that indicates where to find data, how to parse and interpret the data, where to insert the data, and so on. Although not precisely defined, a control file can be said to have three sections.

- The first section contains sessionwide information, including:
 - Global options such as the input data file name and the records to be skipped
 - `INFILE` clauses to specify where the input data is located
 - Data to be loaded

- The second section consists of one or more `INTO TABLE` blocks. Each of these blocks contains information about the table (such as the table name and the columns of the table) into which the data is to be loaded.
- The third section is optional and, if present, contains input data.

Log file: When SQL*Loader begins execution, it creates a log file. If it cannot create a log file, execution terminates. The log file contains a detailed summary of the load, including a description of any errors that occurred during the load.

Bad file: The bad file contains records that are rejected, either by SQL*Loader or by the Oracle database. Data file records are rejected by SQL*Loader when the input format is invalid. After a data file record is accepted for processing by SQL*Loader, it is sent to the Oracle database for insertion into a table as a row. If the Oracle database determines that the row is valid, the row is inserted into the table. If the row is determined to be invalid, the record is rejected and SQL*Loader puts it in the bad file.

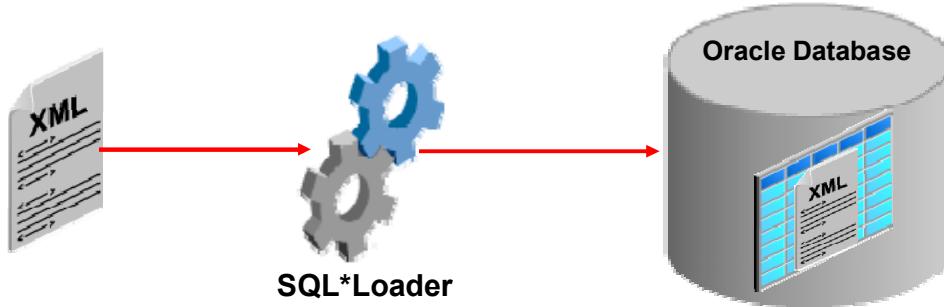
Discard file: This file is created only when it is needed, and only if you have specified that a discard file should be enabled. The discard file contains records that are filtered out of the load because they do not match any record-selection criteria specified in the control file.

For more information about SQL*Loader, refer to the *Oracle Database Utilities* documentation.

Loading XMLType Data by Using SQL*Loader

SQL*Loader:

- Supports loading XMLType data into XMLType tables
- Can load data stored either in a LOB data file or in a control file
- Supports data loading into fields with either predetermined or variable size



ORACLE

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

SQL*Loader supports loading XMLType tables, and the loading is independent of the underlying storage. You can load XMLType data whether it is stored in a large object (LOB) data file or in a control file.

To load internal LOBS, BLOBS, and CLOBS, or XMLType columns and tables from a primary data file, the following standard SQL*Loader formats can be used:

- **Loading LOB data in predetermined size fields:** This is a very fast and conceptually simple format to load LOBS. If the LOBS that you are loading are not of equal sizes, you can pad the LOBS with white spaces to make them of equal length.
- **Loading LOB data in delimited fields:** This format handles LOBS of different sizes, delimited by a specific delimiter string, within the same column. Because SQL*Loader scans through the data looking for the delimiter string, the added flexibility reduces performance.

Loading XMLType Data Stored in a Control File

```
LOAD DATA
INFILE *
TRUNCATE INTO TABLE test_load
XMLType(xmldata)
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ''
(
xmldata
)
BEGINDATA
<EMP><name>KING</name></EMP>,
<EMP><name>SCOTT</name></EMP>,
<EMP><name>SMITH</name></EMP>
```

load_data.ctl

1

```
CREATE TABLE test_load OF XMLType
```

2

```
-- Type the following command on one line at the prompt
command_prompt>sqlldr hr/hr
control=/home/oracle/labs/xml_dir/load_data.ctl
log=aa.log
```

3

ORACLE

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

The following are the steps to load XMLType data stored in a control file by using SQL*Loader:

Step 1: As shown in the slide, create a control file with the extension .ctl and store the XMLType data in it.

Control File Parameters

- **INFILE:** You can use the `INFILE` keyword to specify a data file. “*” specifies that the XML data is to be found in the control file. If the XML data is in an external file, the name of the file containing the data is specified with `INFILE`.
- The `INTO TABLE` statement is required to identify the table that is to be loaded (`test_load`) into.
- `FIELDS TERMINATED BY ',',` specifies that the data is terminated by commas but may also be enclosed within quotation marks.
- `BEGINDATA:` If your data is contained in the control file and not in a separate data file, you must include the data after the `BEGINDATA` keyword. `BEGINDATA` specifies the beginning of the XMLType data. Note the use of commas to delimit multiple records in the specified XMLType data.

Loading Data into Empty Tables

- **INSERT:** This is the default method of SQL*Loader. It requires the table to be empty before loading. SQL*Loader terminates with an error if the table contains rows.

Loading Data into Nonempty Tables

- **APPEND:** If data exists in the table, SQL*Loader appends the new rows to it. If data does not exist, the new rows are simply loaded.
- **REPLACE:** All the rows in the table are deleted and the new data is loaded.
- **TRUNCATE:** This truncates the table (no rollback), and then loads the new data.

Step 2: Create an XMLType table.

Step 3: At the operating system command prompt, execute the `sqlldr` utility by passing the following keywords:

Command-Line Keywords

- **UserId:** This is used to provide your Oracle username and password. If omitted, SQL*Loader prompts you for the username.
- **Control:** This specifies the name of the control file that describes how to load the data. If a file extension or file type is not specified, it defaults to `CTL`. If omitted, SQL*Loader prompts you for the file name.
- **Log:** You can also mention a log file that SQL*Loader will create to store logging information about the loading process. If a file name is not specified, the name of the control file is used, by default, with the default extension (`LOG`).
- **Bad:** You can specify the name of the bad file created by SQL*Loader to store records that cause errors during insert or that are improperly formatted. If a file name is not specified, the name of the control file is used, by default, with the `.BAD` extension.
- **Data:** You can specify the name of the data file that contains the data to be loaded. If a file name is not specified, the name of the control file is used by default. If you do not specify a file extension or file type, the default is `.DAT`.

To confirm whether the data load is successful, you can query the `test_load` table with the following select statement:

```
SELECT * from test_load;
```

SYS_NC_ROWINFO\$

<EMP><name>KING</name></EMP>
<EMP><name>SCOTT</name></EMP>
<EMP><name>SMITH</name></EMP>

Loading XMLType Data Stored in a Separate File

```

<person xmlns="http://www.oracle.com/person.xsd"      person.dat
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oracle.com/person.xsd"
    http://www.oracle.com/person.xsd">
  <name>xyz name 2</name>
</person>
<!-- end of record -->

```

1

```

LOAD DATA
INFILE *
INTO TABLE test_load
APPEND XMLType(XMLDATA)
(
  lobfn FILLER CHAR TERMINATED BY ',',
  XMLDATA LOBFILE(lobfn) TERMINATED BY '<!-- end of record -->'
)
BEGINDATA
/home/oracle/labs/xml_dir/person.dat

```

2

```

command_prompt>sqlldr hr/hr
control=/home/oracle/labs/xml_dir/load_data2.ctl log=aa.log

```

3

ORACLE

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

To use SQL*Loader to load XMLType data stored in a separate file, perform the following:

- Step 1:** Create a file containing XML data. The example in the slide uses the `<!-- end of record -->` delimiting string for terminating the record.
- Step 2:** As shown in the slide, create a control file with the `.ctl` extension. To facilitate loading, the `FILLER` keyword specifies a data file-mapped field that does not correspond to a database column. The `LOBFILE` function is used to specify the name of the field that specifies the name of the file, which contains the data for the `LOB` field. The `lobfn` keyword initializes an empty `LOB` with the `FILLER` characters, and then loads the data. The name of the file (along with its location) associated to the `FILLER` field is specified by using `BEGINDATA`.
- Step 3:** At the command prompt, execute the `sqlldr` utility by passing the control file, username, and the password as parameters.

SYS_NC_ROWINFO\$
1 <EMP><name>KING</name></EMP>
2 <EMP><name>SCOTT</name></EMP>
3 <EMP><name>SMITH</name></EMP>
4 <person xmlns="http://www.oracle.com/person.xsd" xmlns:xsi="http://www.w3....

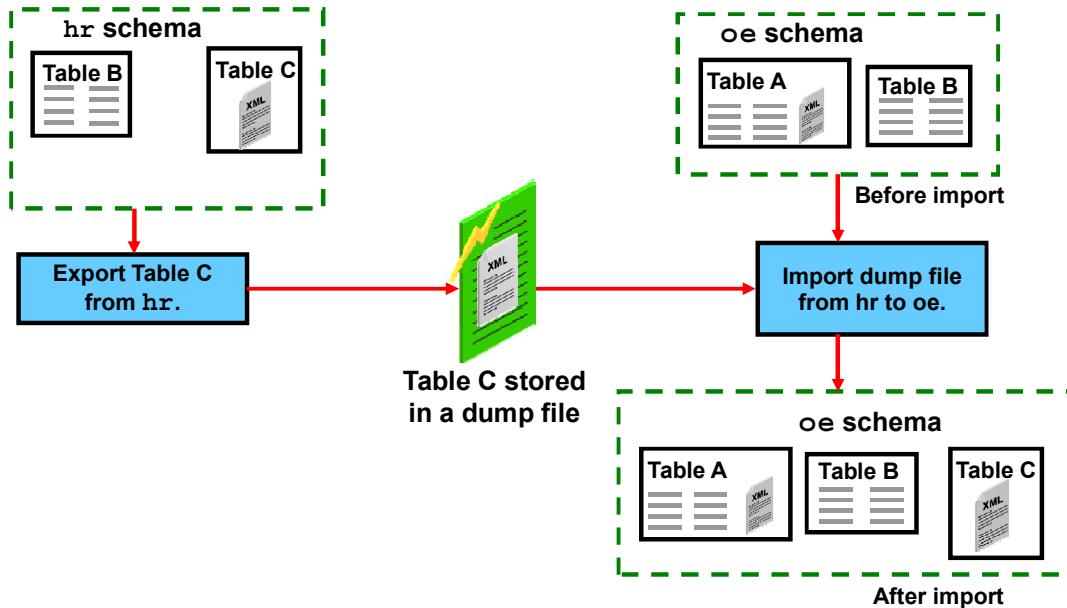
Lesson Agenda

- SQL*Loader
 - Overview
 - Loading XMLType data
- Oracle Data Pump
 - Overview
 - Exporting and importing XMLType data



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

Exporting and Importing XMLType Tables and Columns



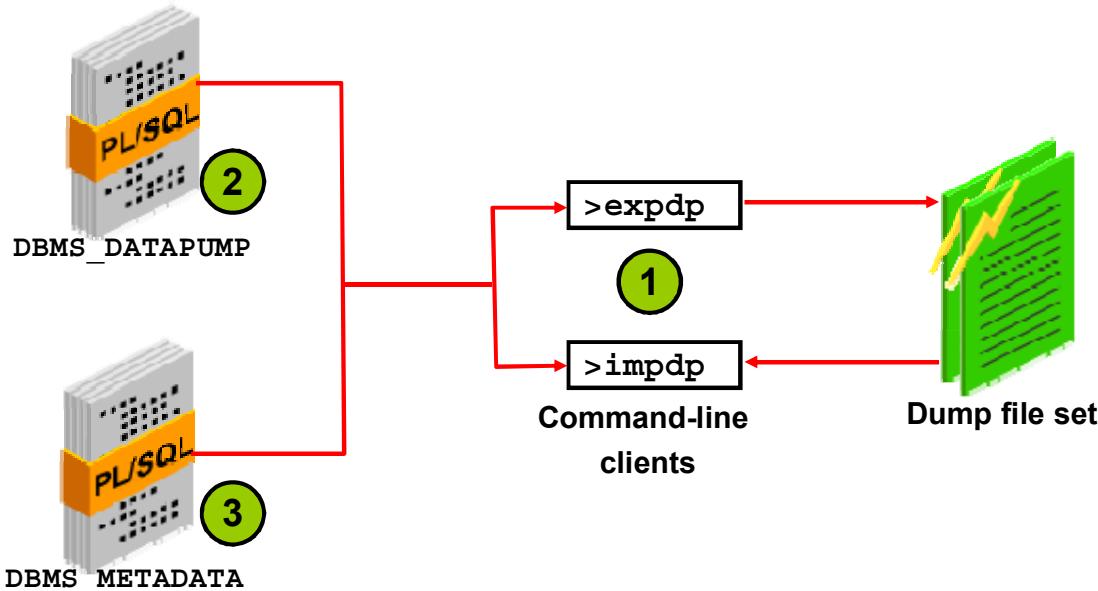
ORACLE

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

The Oracle Data Pump import and export utilities provide a simple way for you to transfer data objects between Oracle databases, even if they reside on platforms with different hardware and software configurations.

Oracle XML DB supports import and export of XMLType tables and columns. You can import and export tables, whether or not they are based on XML schemas. Non-schema-based XMLType tables and columns are imported and exported in a manner similar to the LOB columns.

Oracle Data Pump: Components



ORACLE

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

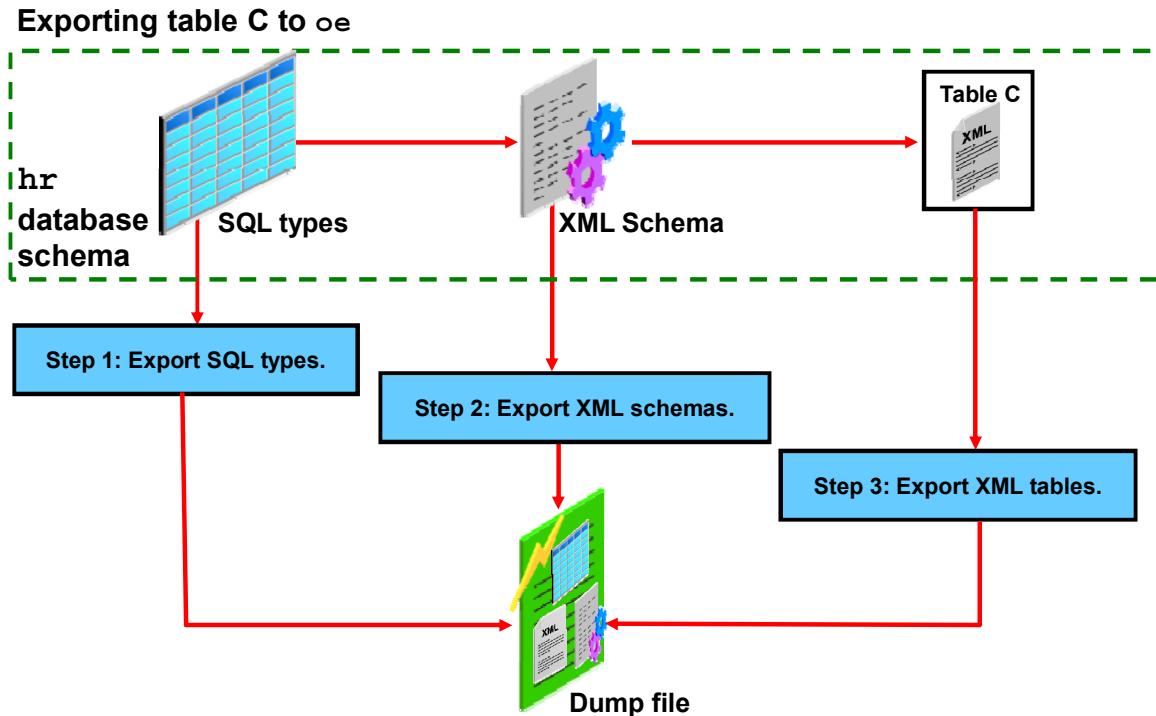
Oracle Data Pump is made up of three distinct parts:

1. The `expdp` and `impdp` command-line clients
2. The `DBMS_DATAPUMP` PL/SQL package (also known as the Data Pump API)
3. The `DBMS_METADATA` PL/SQL package (also known as the Metadata API)

The Data Pump clients, `expdp` and `impdp`, invoke the Oracle Data Pump Export utility and the Oracle Data Pump Import utility, respectively. The `expdp` and `impdp` clients use the procedures that are provided in the `DBMS_DATAPUMP` PL/SQL package to execute the export and import commands by using the parameters entered at the command line. These parameters enable the export and import of data and metadata for a complete database or for subsets of a database. When metadata is moved, Oracle Data Pump uses the functionality provided by the `DBMS_METADATA` PL/SQL package.

Note: The dump files generated by the Oracle Data Pump Export utility are not compatible with the dump files generated by the original Export utility. Therefore, the files generated by the original Export (`expdp`) utility cannot be imported with the Oracle Data Pump Import (`impdp`) utility.

Exporting XML Schema-Based XMLType Tables



ORACLE

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

In Oracle Database, an XMLType table depends on the XML schema that is used to define the table. The schema too has dependencies on the SQL object types that are created or specified for it. Thus, exporting an XML schema-based XMLType table includes the following steps:

Step 1: Export SQL types during XML schema registration.

As part of the XML schema registration process, SQL types can be created. These SQL types are exported as part of the CREATE TYPE statements.

Step 2: Export XML schemas.

After all the SQL types are exported, XML schemas are exported as XML text as part of a call to the DBMS_XMLSCHEMA.registerSchema PL/SQL procedure.

Step 3: Export XML tables.

The next step is to export the tables. Export of each table comprises the following steps:

- The table definition is exported as part of the CREATE TABLE statement.
- The data in the table is exported as XML text. Note that data for out-of-line tables is not explicitly exported. It is exported as part of the data for the parent table.

Note: Oracle Data Pump for Oracle Database 11g, Release 1 (11.1) does not support the export of XML schemas, XML schema-based XMLType columns, or binary XML data to database releases before 11.1.

Export and Import Modes

When exporting or importing database objects, you can specify one of the following modes:

- FULL
- SCHEMAS
- TABLES
- TABLESPACES
- TRANSPORT_TABLESPACE



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

While exporting or importing database objects, you can specify one of the following modes:

- FULL: In a full database export or import, the entire database is unloaded or loaded. This mode requires that you have the EXP_FULL_DATABASE or IMP_FULL_DATABASE role.
- SCHEMAS: This mode is specified to unload or load a list of schemas. If you do not have the EXP_FULL_DATABASE role, you can export only your own schema. In a schema import, only objects owned by the specified schemas are loaded.
- TABLES: In this mode, only a specified set of tables, partitions, and their dependent objects are unloaded or loaded. You must have the EXP_FULL_DATABASE or IMP_FULL_DATABASE role to specify tables that are not in your own schema.
- TABLESPACES: In the TABLESPACES mode, all objects contained in a specified set of tablespaces are unloaded or loaded along with the dependent objects.
- TRANSPORT_TABLESPACE: In this mode, only the metadata for the tables (and their dependent objects) within a specified set of tablespaces is exported or imported.

Exporting XMLType Data: Examples

```
-- This example exports the entire hr schema into  
-- exp_hr_full.dmp.
```

1

```
command_prompt> expdp hr/hr DIRECTORY=xml_dir  
DUMPFILE=exp_hr_full.dmp FULL=y NOLOGFILE=y
```

```
-- This example demonstrates an export in TABLES mode,  
-- which exports only the table-related metadata and  
-- data.
```

2

```
command_prompt> expdp hr/hr TABLES=employees,jobs  
DUMPFILE=xml_dir:exp_hr_table.dmp NOLOGFILE=y
```



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

The examples in the slide are based on the assumption that the `hr` user has the `xml_dir` directory object pointing to a folder in the local file system. A directory object is a name in the database for a physical directory on the host computer.

Example 1 exports the entire `hr` schema into `exp_hr_full.dmp`. You can later import `exp_hr_full.dmp` and obtain all the `hr` objects.

Note: To perform a full export, you must have the `EXP_FULL_DATABASE` role.

Example 2 demonstrates an export in `TABLES` mode, which exports only the table-related metadata and data.

Importing XMLType Data: Example

```
-- This example imports all the data in the  
-- exp_hr_full.dmp file to an hr schema.
```

1

```
command_prompt> impdp hr/hr  
DUMPFILE=xml_dir:exp_hr_full.dmp FULL=y  
LOGFILE=xml_dir:imp_hr_full.log
```

```
-- This example imports only the employees and jobs  
-- tables from the exp_hr_full.dmp file.
```

2

```
command_prompt> impdp hr/hr DIRECTORY=xml_dir  
DUMPFILE=exp_hr_full.dmp TABLES=employees,jobs
```



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

Example 1 imports all the data in the `exp_hr_full.dmp` file to an `hr` schema. The user that you specify during the import must exist in the target database before the import operation. Otherwise, an error is returned. The import also fails if source object types and object tables exist on the target system.

If your source schema and target schema are different, you must use the `REMAP_SCHEMA` parameter in your command. For example, if you want to import a table from an `hr` schema to an `oe` schema, you must specify the following command:

```
impdp oe/oe DIRECTORY=xml_dir DUMPFILE=exp_hr_full.dmp  
TABLES=employees REMAP_SCHEMA=oe:hr
```

If you have read access to a dump file, you can import it whether or not you are the user who exported the data into it. However, if you are not the exporter of the data into the dump file, to import it, you must also have the `IMP_FULL_DATABASE` role granted to you.

Note: The `exp_hr_full.dmp` file that is used in the example is obtained by exporting objects from the `hr` schema as shown in the code example in the preceding slide.

Example 2 imports only the `employees` and `jobs` tables from the `exp_hr_full.dmp` file.

Quiz

SQL*Loader can load data stored in a LOB data file but not in a control file.

- 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:

- Load XMLType data by using SQL*Loader
- Import and export XMLType data by using Oracle Data Pump



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

Practice 16: Overview

This practice covers the following topics:

- Using the SQL*Loader utility to load XMLType data stored in the control file or in an external data file
- Exporting data stored in a user's schema and importing it into another user's schema by using a dump file



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

17

Workshop

ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Store XML data in Oracle XML DB
- Create folders and resources in Oracle XML DB Repository
- Access resources in Oracle XML DB Repository
- Use SQL/XML standard functions and XQuery to retrieve the XML data that is stored in Oracle XML DB
- Create the `XMLIndex` index and improve the performance of XQuery expressions



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

In Workshop I, you load library data into Oracle XML DB. You then create indexes and execute the SQL/XML statements with XQuery expressions.

In Workshop II, you create resources in Oracle XML DB Repository. You create tables and load tourism data. Finally, you use XQuery to retrieve data from Oracle XML DB.

Prerequisites

Before starting the two workshops, make sure that you have successfully completed all the practices for this course.



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

Workshop I

In this workshop, you perform the following:

- Verify SQL Developer preferences
- Load the library data file into Oracle XML DB
- Create a table with an XMLType column by using the binary XML storage and load library data
- Create indexes to improve the performance of XQuery expressions
- View the execution plans of the queries
- Perform DML operations



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

You are a database developer and you help maintain a database for the Library department. You have to work with the following data:

- Library articles, journals, dates, and other references in `library_info.xml`
- Library information inserted into the `library_info_tab` table that is created with an `XMLType` column by using the binary XML storage

First, you verify the SQL Developer preferences. You must set the Autotrace parameters so that you can restrict the display of information when you view the execution plans in SQL Developer.

In this workshop, you create `library_info_tmp_tab`, a temporary table with an `XMLType` column by using the binary XML storage. The column name can be `citations`. Then, load the library data into `library_info_tmp_tab`.

In the next step, you create the `library_info_tab` table with an `XMLType` column by using the binary XML storage. The column name can be `citation`. Then insert the library data from the temporary table `library_info_tmp_tab` into the `library_info_tab` table.

You then create an `XMLIndex` index on the binary XML column and a secondary Full-Text index on the value of the `XMLIndex` index. In the next step, you run the SQL/XML statements with the XQuery expressions. Then, you observe the explain plan to see how the query and the data manipulation language (DML) executions take advantage of the `XMLIndex` index.

For this workshop, you must be connected to the database as hr with the password hr.

Verifying SQL Developer Preferences

1. Verify SQL Developer preferences.
 - a. Go to **Tools > Preferences**.
 - b. Expand Database and select **Autotrace Parameters**.
 - c. Make sure to select the following check boxes and click **OK**.
 - Object_Name
 - Cost
 - Cardinality

Loading Library Data into Binary XML Storage

2. Create a temporary table by using the binary XML storage to load the library data. The table must contain a column `citations`. The `citations` column must be of `XMLType`. The table can be called `library_info_tmp_tab`.
3. By using `lab_load_data1.sql`, insert the library data from `library_info.xml`.
4. Create another table with an `XMLType` column by using the binary XML storage to store all the citations from the library data. The table must contain a column `citation`. The `citation` column must be of `XMLType`. The table can be called `library_info_tab`.
5. By using `lab_load_data2.sql`, insert the library data from the `library_info_tmp_tab` temporary table.

Improving the Performance of XQuery Expressions Through Index Creation

6. Open `lab_cre_index1.sql`. Replace `<TODO>` with the appropriate code to create an `XMLIndex` index on the `citation` column of the `library_info_tab` table. The index can be called `citation_bix_ix`. Save the changes, and then run `lab_cre_index1.sql`.
7. Create a secondary Full-Text index on the `VALUE` column of the path table. Open `lab_cre_index2.sql`. Replace `<TODO>` with the appropriate code, save the changes, and then run `lab_cre_index2.sql`.
8. By using `XMLExists`, write a query on the `library_info_tab` table to display the number of citations whose `DateCreated` year is greater than or equal to 2006. Use `lab_query1.sql` for help.
9. By using `XMLTable` and `XMLExists`, write a query on the `library_info_tab` table to display the list of article titles for all those citations whose `DateCreated` year is greater than or equal to 2006. Use `lab_query2.sql` for help.
10. In the SQL Worksheet, select the entire code from `lab_query2.sql` and click the **Autotrace** icon. Observe the Autotrace output pane and note that the execution plan uses the `XMLIndex` index.
11. By using `XMLExists`, write a query on the `library_info_tab` table to display the number of citations for which `ArticleTitle` contains the text "Training." Use `lab_query3.sql` for help.
12. By using `XMLTable` and `XMLExists`, write a query on the `library_info_tab` table to display the list of article titles for all those citations where `ArticleTitle` contains the text "Training." Use `lab_query4.sql` for help.

Performing DML Operations

13. In the SQL Worksheet, open `lab_dml_query1.sql`. Click **Execute Statement**.
14. Update the ID value 12332017 to 12332017_a. Use `lab_dml.sql` for help. Save the changes and run `lab_dml.sql`.
15. In the SQL Worksheet, select the entire code in `lab_dml.sql` and click the **Autotrace** icon. Observe the Autotrace output pane and note that the execution plan uses the XMLIndex index.
16. Verify that the ID value is updated. Run `lab_dml_query2.sql`.

Workshop II

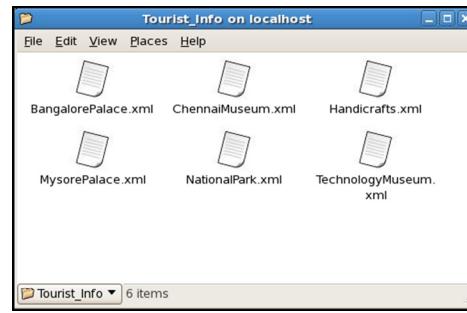
TOURIST_DIVISION table with data related to all tourist divisions

DESCRIBE TOURIST_DIVISION		
Name	Null	Type
TD_ID		NUMBER
TD_NAME		VARCHAR2(50)
TD_PHONE		VARCHAR2(15)
TD_REGION		VARCHAR2(15)

DESCRIBE REGION_COUNT		
Name	Null	Type
RC_REGION		VARCHAR2(15)
RC_CITY_POPULATION		XMLTYPE()

REGION_COUNT table with an XMLType column. Each row contains an XML document with population counts for all the regions in a division.

Information about tourist attractions in XML documents that you access by using Oracle XML DB Repository



ORACLE

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

You are a database developer and you help maintain a database for the tourism industry. You must work with the following data:

- Information about tourist attractions in XML documents that you access from Oracle XML DB Repository:
 - BangalorePalace.xml
 - ChennaiMuseum.xml
 - Handicrafts.xml
 - MysorePalace.xml
 - NationalPark.xml
 - TechnologyMuseum.xml
- Information about tourist divisions that is stored in the TOURIST_DIVISION relational table
- Population data as documents in an XMLType column in the REGION_COUNT table

The definition of the TOURIST_DIVISION and REGION_COUNT tables that you create should look like the output of the respective DESCRIBE statements in the slide.

Workshop II

In this workshop, you generate the following:

- A set of region nodes from the `TOURIST_DIVISION` table
- Data from the `TOURIST_DIVISION` table and a transformation of the data
- Population counts from the XML documents in the `REGION_COUNT` table



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

First, you create the `Tourist_Info` Repository folder under `/public`. Then you create the resources (XML files) in the `/public/Tourist_Info` folder. When you use FTP and access Oracle XML DB Repository, the folder structure should look similar to the one shown in the previous slide.

In this workshop, you must generate the following:

- Set of region nodes from the `TOURIST_DIVISION` table
- Data from the `TOURIST_DIVISION` table. You must then transform the data. The output should appear in the following style:

```
<regions><region><name>region name</name><destination phone="Tourist Division Phone">Tourist Division Name</destination></region></regions>
```
- Population counts from the XML documents in the `REGION_COUNT` table

For this workshop, you must be connected to the database as hr with the password hr.

1. Run lab_del_resources.sql. This script deletes existing resources (if any).
2. You need a folder in Oracle XML DB Repository to store all the information about the tourist attractions in XML files. By using DBMS_XDB, create an anonymous block to create a folder Tourist_Info under /public. Use lab_cre_resources1.sql for help. Replace <TODO> with the appropriate code, save the changes, and then run lab_cre_resources1.sql.
3. By using FTP, view the folder structure that was created in step 2.
4. Now that you have the Tourist_Info folder, you need to create XML files for each tourist attraction. By using DBMS_XDB, create resources in the /public/Tourist_Info folder. The Tourist_Info folder should contain the following files:
 - BangalorePalace.xml
 - TechnologyMuseum.xml
 - MysorePalace.xml
 - NationalPark.xml
 - ChennaiMuseum.xml

You may use lab_cre_resources2.sql for help. Replace <TODO> with the appropriate code to create the respective Repository resources and save the file. Run lab_cre_resources2.sql.

5. By using FTP, view the folder structure that was created in step 4.
6. You need a relational table to store the information related to all the tourist divisions. You want to store the tourist division ID, division name, telephone number, and the name of the main region. Create a table TOURIST_DIVISION and insert data into it. Run lab_cre_td_tab.sql.
7. Create a table to store the population data of the regions. The table can be called REGION_COUNT. This table should contain two columns: rc_region VARCHAR(15) and rc_city_population (as an XMLType column).
8. Insert the population data into the REGION_COUNT table. Run lab_load_rc_tab.sql.
9. By using XMLQuery, generate a set of region nodes from the TOURIST_DIVISION table. Use lab_xquery1.sql for help. Replace <TODO> with the appropriate code, save the changes, and then run lab_xquery1.sql. Click the **Run Script** icon or press **F5** in SQL Developer.
10. By using XMLQuery, generate data from the TOURIST_DIVISION table. Then transform the data. The output should appear as follows:

```
<regions><region><name>Bangalore</name><destination phone="" (80) 2663 2142">Bangalore Tourism Corp</destination></region></regions>
```

Use lab_xquery2.sql for help. Replace <TODO> with the appropriate code, save the changes, and then run lab_xquery2.sql. Click **Run Script** or press **F5** in SQL Developer.
11. By using XMLQuery, generate the sum of population in each region from the REGION_COUNT table. Use lab_xquery3.sql for help. Replace <TODO> with the appropriate code, save the changes, and then run lab_xquery3.sql. Click **Run Script** or press **F5** in SQL Developer.

Summary

After completing the workshops, you should have learned how to:

- Store XML data in Oracle XML DB
- Create folders and resources in Oracle XML DB Repository
- Access resources in Oracle XML DB Repository
- Use the SQL/XML standard functions and XQuery to retrieve the XML data stored in Oracle XML DB
- Create the `XMLIndex` index and improve the performance of XQuery expressions



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

18

Case Study



ORACLE

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

Objectives

After completing this case study, you should be able to:

- Review the steps and scripts required to set up the environment for the case study
- Create a new database connection for the user for this case study, and check the Autortrace parameters
- Access XML content by using XQuery
- Use XMLIndex, path-subsetted XMLIndex, and structured XMLIndex to optimize XQuery performance
- Use XQuery-Update to update XML content
- Create an XML-aware full-text index and perform XQuery Full Text searches on XML content
- Optimize XML storage with XML schema



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

In this case study, you apply what you learned so far in this course to perform the tasks listed in the Objectives slides.

Objectives

- Use the same XQuery queries that are used with binary XML storage with object-relational storage
- Index object-relational storage to optimize XQuery operations
- Equi-partition object-relational storage
- Create relational views from XML content
- Use SQL/XML functions to generate XML documents from relational tables
- Create a persistent XML view of relational data



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

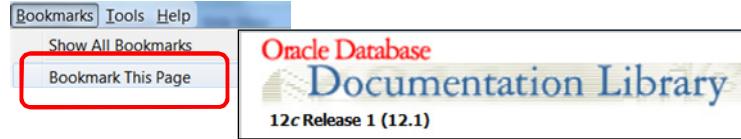
Prerequisites

Before starting this case study, make sure that you have:

- Successfully completed all the practices and workshops in this course
- Accessed and bookmarked the appropriate documentation referenced in this course



Oracle Database 12c: Use XML DB

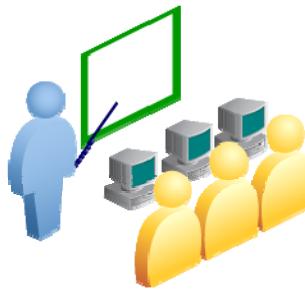


ORACLE

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

Case Study Username and Password Account Information

- Your account username is XDBOE and your password is oracle.
 - When you access the XML DB repository, using WebDAV or a web browser, the username (XDBOE) and password (oracle) are case-sensitive
 - When you use SQL*Plus or SQL Developer, the username is not case-sensitive but the password is case-sensitive.
- Each machine has the required setup for the case study.



ORACLE

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

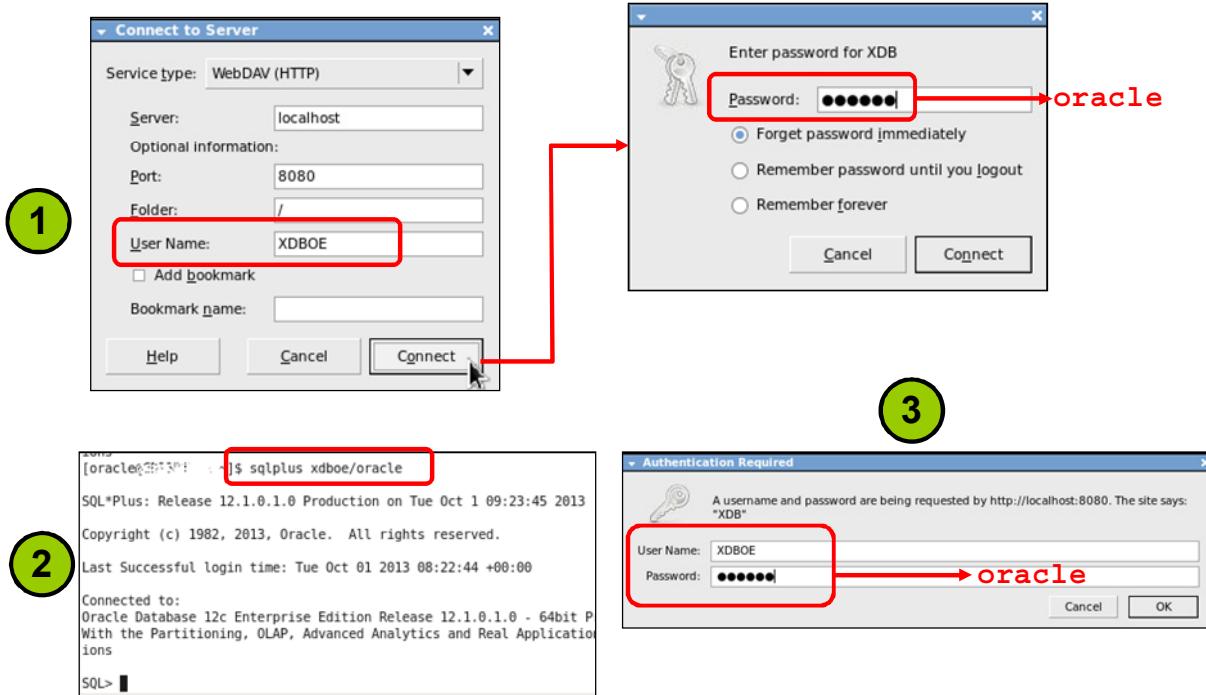
As you have learned earlier in this course, users can now access Oracle XML DB Repository by using digest access authentication (also known as digest authentication), in addition to basic authentication. This provides encryption of user credentials (name, password, and so on) without the overhead of complete data encryption.

Starting with Oracle 12c, user credentials are now case-sensitive. In particular, a username to be authenticated must exactly match the name as it was created (which by default is all uppercase).

In this case study:

- If you are accessing the XML DB repository, the username and password are case-sensitive: The username is XDBOE (uppercase) and the password is oracle (lowercase).
- If you are using SQL*Plus or SQL Developer, the username and password are not case-sensitive.
- If you are using a web browser, the username and password are case-sensitive: The user name is XDBOE (uppercase) and the password is oracle (lowercase).

Case Study Username and Password Account Information

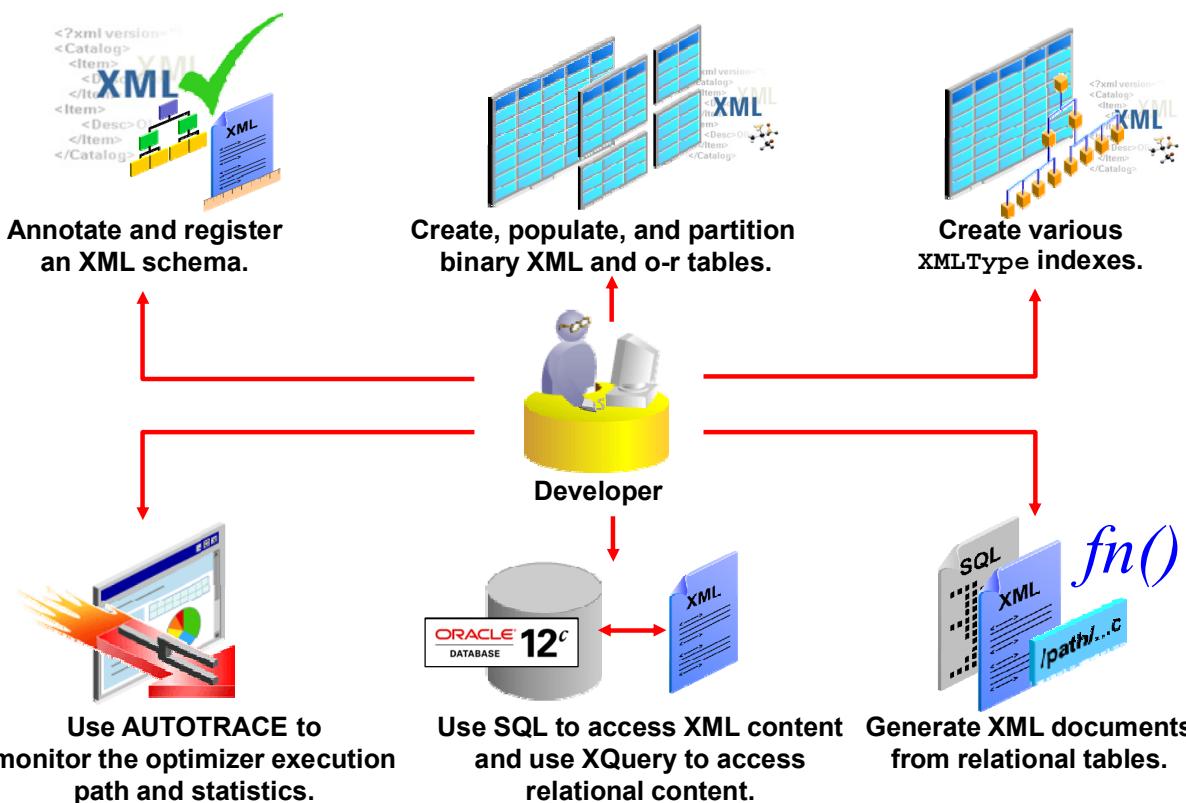


ORACLE

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

1. If you are using the XML DB repository, use XDBOE (uppercase) for the username and oracle (lowercase) for the password.
2. If you are using SQL*Plus or SQL Developer, the username and password are not case-sensitive.
3. If you are using a web browser, use XDBOE (uppercase) for the username and oracle (lowercase) for the password.

Performing XML DB Tasks as a Developer

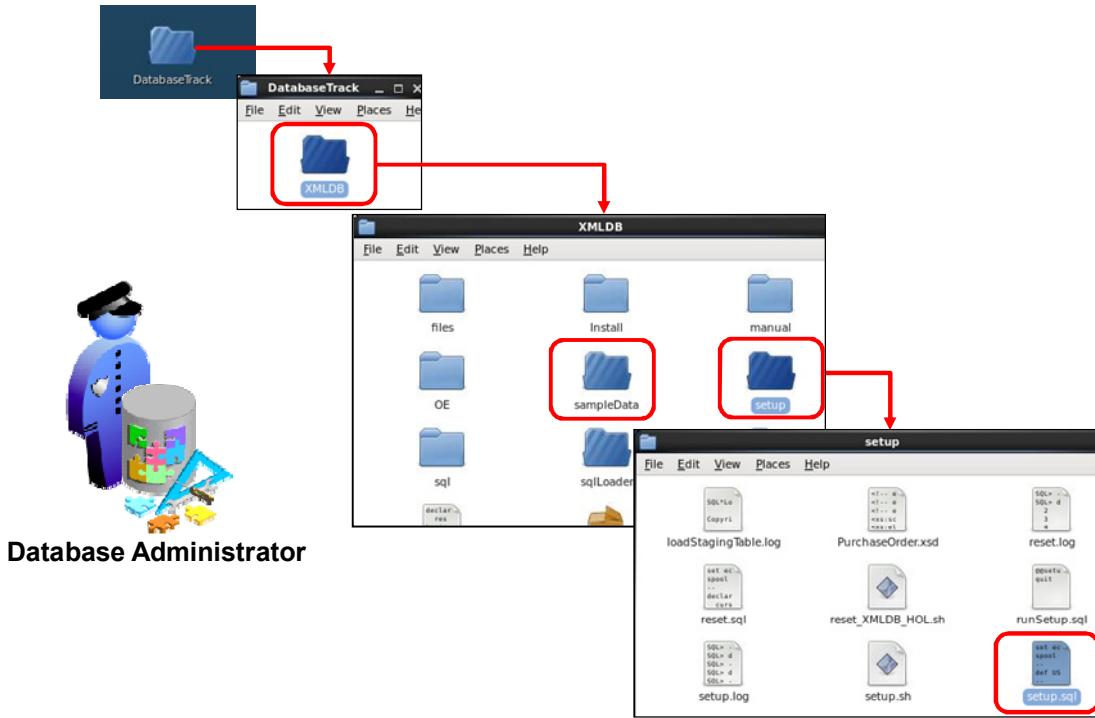


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

In this case study, you are a developer using XML DB and you are in charge of maintaining a database of the purchase orders in your company. The slide displays some tasks that you perform in this role. You create the necessary XML queries and structures in order to support the end users' need for queries. The queries are used against a new `purchaseorder` table that contains 10,000 XML documents containing purchase orders generated by a home products ordering and billing application.

Case Study Setup Files: DatabaseTrack Folder



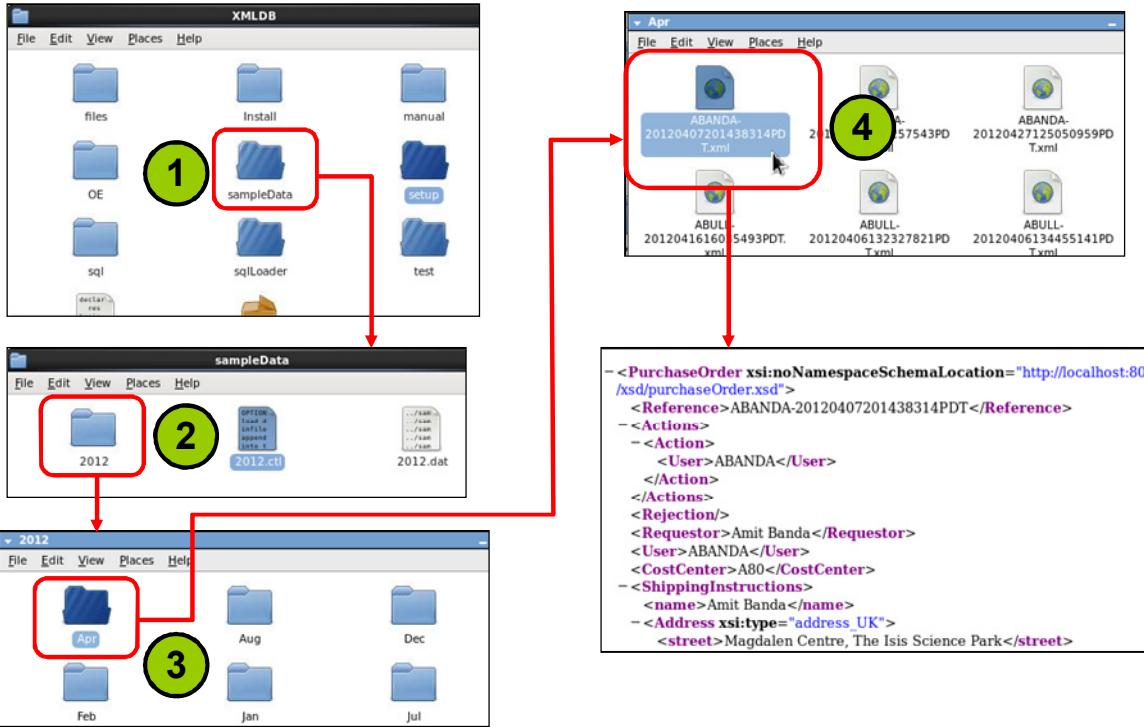
ORACLE

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

Your database administrator has already set up your environment and created the necessary database structures that you will need in this case study.

The DatabaseTrack folder on your desktop contains scripts and other files that are used to create the setup for this case study. The screenshots in the slide show some of the folders, such as the setup folder that contains most of the case study setup scripts.

Case Study Setup Files: sampleData Folder

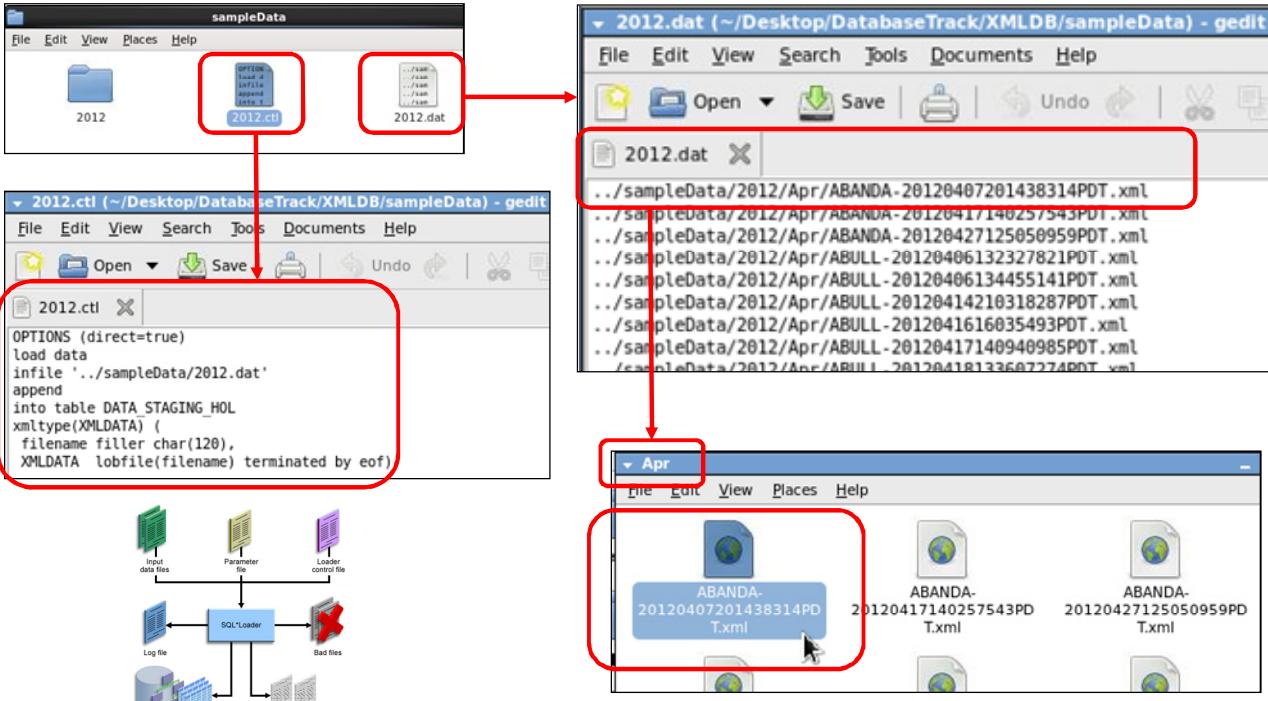


ORACLE

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

1. The sampleData folder in the DatabaseTrack > XMLDB folder contains sample data files that will be used in the case study. It also contains the 2012.ctl control file (shown on the next page) and the 2012.dat file (also shown on the next page) that were used in SQL*Loader to load the 10,000 XML documents into a staging table named DATA_STAGING_HOL. One of your first tasks in this case study is to create a new purchaseorder table and to populate this table from the DATA_STAGING_HOL table.
2. The 2012 folder contains monthly subfolders, which in turn contain the XML documents.
3. For example, the Apr folder contains the XML documents for the month of April. If you double-click the folder, the XML documents are displayed.
4. If you double-click an XML document, it is displayed as shown in the slide.

Case Study Setup Files: sampleData Folder

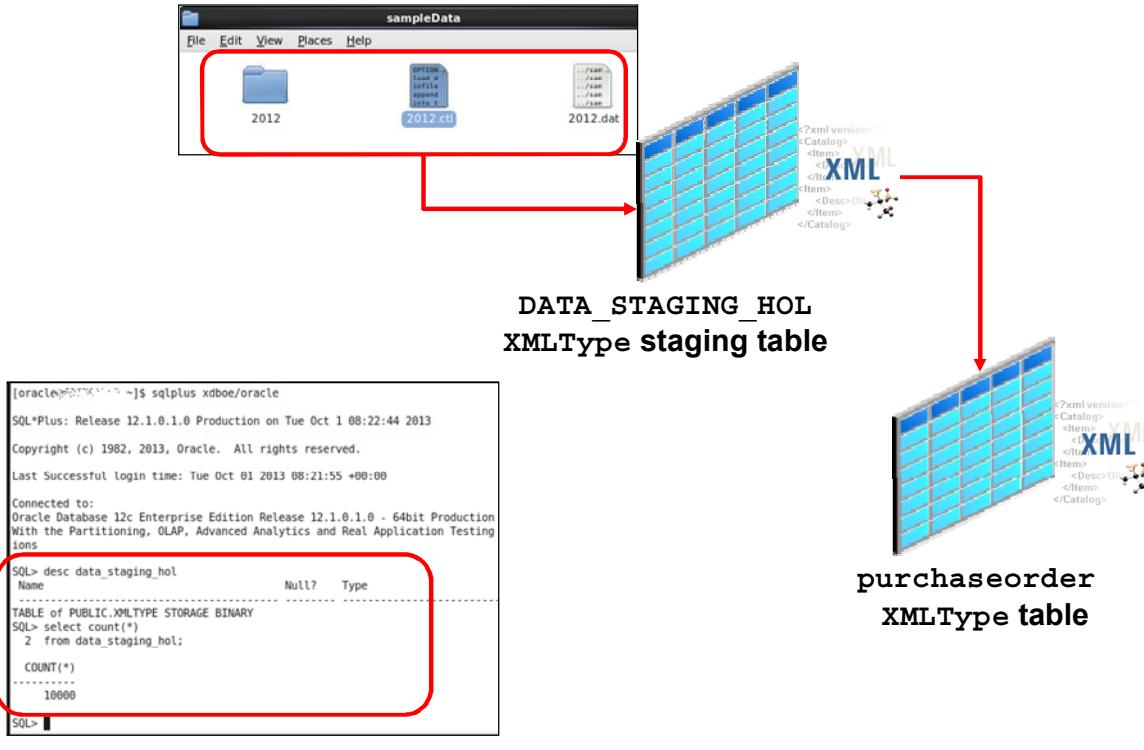


ORACLE

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

The **sampleData** folder also contains the **2012.ctl** control file and the **2012.dat** file that were used in SQL*Loader to load the 10,000 XML documents into a staging table named **DATA_STAGING_HOL**.

DATA_STAGING_HOL XMLType Table



ORACLE

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

As explained on the previous page, the database administrator has already used SQL*Loader to populate the DATA_STAGING_HOL staging table with the data referenced in the 2012.dat file. This file points to the actual XML documents that are found in the 2012 folder.

purchaseorder XMLType Table

The screenshot shows the XMLSpy interface with the file `PurchaseOrder.xsd` open. A red arrow points from the schema definition area to a sample XML document titled `SYS_NC_ROWINFO$`.

```

<!-- edited with XMLSpy v2010 rel. 2 (x64) (http://www.altova.com) by Mark D Drake (Oracle XML DB) -->
<!-- edited with XMLSPY v2004 rel. 2 U (http://www.xmlspy.com) by Mark D. Drake (Oracle XML DB) -->
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema" version="1.0">
  <xss:element name="PurchaseOrder" type="PurchaseOrderType"/>
  <xss:complexType name="PurchaseOrderType">
    <xss:sequence>
      <xss:element name="Reference" type="ReferenceType"/>
      <xss:element name="Actions" type="ActionsType"/>
      <xss:element name="Rejection" type="RejectionType" minOccurs="0"/>
      <xss:element name="Requestor" type="RequestorType"/>
      <xss:element name="UserType" type="UserType"/>
      <xss:element name="CostCenter" type="CostCenterType"/>
      <xss:element name="ShippingInstructions" type="ShippingInstructionsType"/>
      <xss:element name="SpecialInstructions" type="SpecialInstructionsType"/>
      <xss:element name="LineItems" type="LineItemsType"/>
    </xss:sequence>
  </xss:complexType>
  <xss:complexType name="LineItemsType">
    <xss:sequence>
      <xss:element name="LineItem" type="LineItemType" maxOccurs="unbounded">
        <xss:sequence>
          <xss:element name="Part" type="PartType"/>
          <xss:element name="Quantity" type="QuantityType"/>
        </xss:sequence>
      </xss:element>
    </xss:sequence>
  </xss:complexType>
  <xss:complexType name="LineItemType">
    <xss:sequence>
      <xss:element name="Part" type="PartType"/>
      <xss:element name="Quantity" type="QuantityType"/>
    </xss:sequence>
  </xss:complexType>
  <xss:attribute name="ItemNumber" type="xs:integer"/>
</xss:complexType>
<xss:complexType name="PartType">
  <xss:extension base="UPCCodeType">
    <xss:attribute name="Description" type="DescriptionType" use="required"/>
    <xss:attribute name="UnitPrice" type="moneyType" use="required"/>
  </xss:extension>
</xss:complexType>

```

```

SYS_NC_ROWINFO$

<PurchaseOrder>
<Actions>
<User>ACABRIO</User>
</Actions>
<Rejection/>
<Requestor>James Marlow</Requestor>
<User>JAMRLOW</User>
<CostCenter>A50</CostCenter>
<ShippingInstructions>
<name>James Marlow</name>
<Address xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="address_US">
  <street>200 Sporting Green</street>
  <city>South San Francisco</city>
  <state>CA</state>
  <zipCode>99236</zipCode>
  <country>United States of America</country>
</Address>
<telephone>36-357-8363</telephone>
</ShippingInstructions>
<SpecialInstructions>Priority Overnight</SpecialInstructions>
<LineItem>
<LineItem ItemNumber="1">
  <Part Description="Stalag 17" UnitPrice="19.95">97360581645</Part>
  <Quantity>9</Quantity>
</LineItem>
<LineItem ItemNumber="2">
  <Part Description="Family Plot" UnitPrice="19.95">25192065927</Part>
  <Quantity>7</Quantity>
</LineItem>
<LineItem ItemNumber="3">
  <Part Description="Silly Symphonies" UnitPrice="27.95">786936158212</Part>
  <Quantity>9</Quantity>
</LineItem>
<LineItem ItemNumber="4">
  <Part Description="Dragon Ball Z: Androids- Assassins" UnitPrice="27.95">84400030093</Part>
  <Quantity>3</Quantity>
</LineItem>
<LineItem ItemNumber="5">

```

The `purchaseorder` table
conforms to the
`PurchaseOrder.xsd` XML
schema document.

ORACLE

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

The `purchaseorder` table in this case study XDBOE schema contains 10,000 XML documents, whereas the `purchaseorder` table that you used in the oe schema contained 132 XML documents.

The `PurchaseOrder.xsd` XML schema document to which the `purchaseorder` table in the case study conforms is also different from the `PurchaseOrder.xsd` XML schema document that you used earlier in this course.

Case Study Practices: Overview

1. Create a new database connection and set the Autotrace parameter in SQL Developer.
2. Create a new PURCHASEORDER XMLType table, populate it from the DATA_STAGING_HOL XMLType table, and then gather some statistics.
3. Use XQuery, XMLTable, and XMLSERIALIZED to query the PURCHASEORDER table. Examine the execution plans for XQuery expressions.
4. Create an XMLIndex on the PURCHASEORDER table to optimize your XQuery query performance.
5. Create a path-subsetted XML index on the PURCHASEORDER table to optimize your XQuery query performance.



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

In Practice 1, you create a new database connection for the XDBOE/oracle user, connect to the new schema, and then set the Autotrace parameters.

In Practice 2, you create an XMLType table in the XDBOE schema and then load XML documents into the newly created table. Finally, you gather some statistics and then display the new table in SQL Developer.

In Practice 3, you use XQuery and the XMLTable, and XMLSERIALIZED SQL/XML functions to access the XML content from the purchaseorder table.

In Practice 4, you create an unstructured XMLIndex on the PURCHASEORDER table named PURCHASEORDER_IDX to improve the performance of your XQuery operations. Next, you use the dbms_stats.gather_schema_stats procedure to gather some statistics.

In Practice 5, you create a path-subsetted XMLindex on the PURCHASEORDER table to optimize your XQuery performance. A path-subsetted XMLindex narrows the focus of indexing by pruning the set of XPath expressions (paths) corresponding to XML fragments to be indexed, specifying a subset of all possible paths. This reduces both the size of the index and the overhead associated with index maintenance, by indexing only those nodes that will be referenced in predicates.

Case Study Practices: Overview

6. Create a structured XMLIndex on the PURCHASEORDER table to optimize your XQuery query performance.
7. Use XQuery-Update to update the PURCHASEORDER table.
8. Create an XML-aware full-text index script and perform XQuery full-text searches on the PURCHASEORDER table.
9. Use XML schema to optimize XML storage and processing. You view the PurchaseOrder.xsd XML schema document using a web browser. You also annotate and register the XML schema.
10. Confirm that no changes are required to execute the XQuery expressions used with the binary XML table on a new object-relational table.



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

In Practice 6, you create a structured XMLIndex to optimize your XQuery performance. With a structured XMLIndex, one or more XMLTable operators are used to define which nodes in the document will be indexed. The selected nodes are projected into a series of relational tables that form the basis of the index. When the index has been created, the system identifies the XQuery expressions that can be evaluated using the index, and uses the index appropriately. You can create secondary indexes on tables that make up the index, enabling further performance optimizations. You also use Autotrace to generate the execution plan for some XQuery queries to determine whether or not the XMLIndex is used.

In Practice 7, you use XQuery-Update to update the PURCHASEORDER table. You use the XMLQuery() SQL/XML function to perform XQuery-Update operations on XML content. You also use the XMLExists SQL/XML function to determine the documents to which the XQuery-Update operation is applied.

In Practice 8, you create an XML-aware full-text index. You then perform XQuery Full Text searches on the XML content stored in the PURCHASEORDER table.

In Practice 9, you use an XML schema to optimize storage, index, query, and manage structured XML data. You also view the PurchaseOrder.xsd XML schema document by using a web browser. You also annotate and register the XML schema.

In Practice 10, you confirm that no changes are required to execute the XQuery expressions used with the binary XML table on a new object-relational table.

Case Study Practices: Overview

11. Create indexes on object-relational storage to optimize XQuery .



12. Create equi-partitioning on object-relational storage to allow partition pruning and partition maintenance operations.



13. Use XQuery and XMLTable to enable efficient SQL relational access and operations on XML content.



14. Use XQuery to generate XML documents from relational tables.



15. Create a persistent XML view of relational data.



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

In Practice 11, you create indexes to optimize queries on the Reference and User nodes and a compound index on the repeating elements Part Number and Quantity. You use the XPATH2TABCOLMAPPING procedure in the DBMS_XMLSTORAGE_MANAGE package to determine which table and column correspond to a particular XPath expression. Next, you use this information to generate a DDL statement that creates the appropriate index.

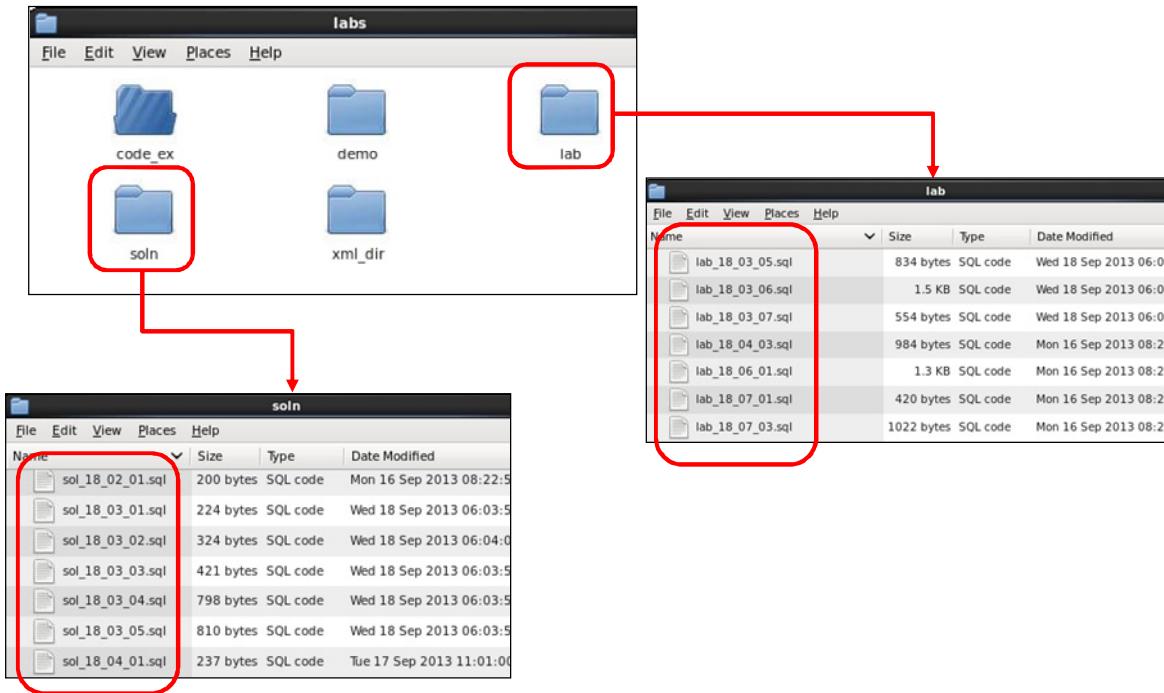
In Practice 12, you create a partitioned object-relational XMLType table. Partitioning helps simplify XML data lifecycle management and performance. Equi-partitioning ensures that all of the nested tables used to manage collections automatically follow the partitioning strategy defined for the parent XMLType table. This allows partitioning pruning and partition maintenance operations to operate on XMLType tables in the same way as they operate on simple relational tables.

In Practice 13, you use the combination of XQuery and XMLTable to enable efficient SQL operations on XML content.

In Practice 14, you use XQuery to generate XML documents from relational tables. You also use SQL/XML functions to generate XML documents from relational tables.

In Practice 15, you create an XML view of relational data. The view is a view of XMLType, which means that the view definition must include a mechanism for generating a unique ID for each row in the view.

Case Study Script Locations: lab and soln Folders

**ORACLE**

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

The practices and solutions to the case study 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. Example:

The `sol_18_02_01.sql` script is the solution code for question 1 in Practice 18-2 in the case study.

The `lab_18_03_05.sql` script is the practice used in question 5 in Practice 18-3.

Additional Information in This Course

Lesson Number and Name	Topics Related to the Case Study
Lesson 4, “Storing XML Data in Oracle XML DB”	<ul style="list-style-type: none"> • Create XMLType tables and columns. • Load data into XMLType columns.
Lesson 5, “Using XML Schema with Oracle XML DB”	<ul style="list-style-type: none"> • Create XML schema-based XMLType tables. • Delete and register an XML schema. • Use XML schema evolution to manage changes in an XML schema. • Perform copy-based and in-place XML schema evolution.
Lesson 6, “Oracle XML DB Manageability”	Annotate an XML schema by using the PL/SQL package DBMS_XMLSHEMA_ANNOTATE.
Lesson 7, “Partitioning XMLType Tables and Columns”	Partition XMLType tables and columns stored object relationally.
Lesson 8, “Using XQuery to Retrieve XML Data in Oracle XML DB”	<ul style="list-style-type: none"> • 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.

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

The tables on this page and the next two pages list some of the lessons and topics that you can use a reference as you work on the case study. Not all the topics in the listed lessons are listed in the tables.

Additional Information in This Course

Lesson Number and Name	Topics Related to the Case Study
Lesson 9, “Updating XML Content Using XQuery Update”	<ul style="list-style-type: none">• Update an entire XML document.• Replace and delete XML nodes.• Insert an element into a collection.
Lesson 10, “Searching XML Content Using XQuery Full Text”	<ul style="list-style-type: none">• Create the supporting Oracle Text structures required to create a full-text index.• Create a full-text index.• Use XQuery Full Text queries that use the full-text index.
Lesson 11, “Indexing XMLType Data”	<ul style="list-style-type: none">• Create an XMLType index.• Specify paths for XMLIndex.• Create an XMLIndex index with structured component.



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

Additional Information in This Course

Lesson Number and Name	Topics Related to the Case Study
Lesson 12, “Generating XML Data”	Use XQuery, standard SQL/XML functions, Oracle SQL functions to generate XML, and the DBMS_XMLGEN PL/SQL package to generate XML.
Lesson 13, “Transforming and Validating XMLType Data”	Transform and validate XML data.
Lesson 14, “Working With the Oracle XML DB Repository”	<ul style="list-style-type: none">• Create folders and resources in the repository.• Use the RESOURCE_VIEW and PATH_VIEW APIs to access the repository.• Use access control lists.
Lesson 16, “Exporting and Importing XML Data”	<ul style="list-style-type: none">• Load XMLType data by using SQL*Loader.• Export and import XMLType data by using Oracle Data Pump.



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

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.

Practice 18-1: Creating a New Database Connection and Setting the Autotrace Parameters

In this practice, you create a new user XDBOE with the password oracle. Next, you verify and set some SQL Developer preferences. You use the new user and password throughout the case study.

1. Click the SQL Developer icon on the desktop to start the application.
2. Create a new database connection.
3. Enter the following details in the New>Select Database Connection window:
 - a. Connection Name: XDBOE
 - b. Username: XDBOE
 - c. Password: oracle
 - d. Save Password: Enable
 - e. Hostname: localhost
 - f. SID: orcl
4. Test the connection, and then connect to the database using the new database connection.
5. Verify the SQL Developer Autotrace/Explain Plan parameter preferences.
 - a. Go to Tools > Preferences.
 - b. Expand the Database node, and then select Autotrace/Explain Plan parameter.
 - c. Select the following check boxes, if not already selected, and then click OK:
 - Object_Name
 - Cost
 - Cardinality
 - Predicates

Practice 18-2: Creating an XMLType Table and Loading Data

In this practice, you create an XMLType table in the XDBOE schema and then load XML documents into the newly created table. Finally, you gather some statistics and then display the new table in SQL Developer.

1. Use the `lab_18_02_01.sql` script to create a simple binary XML XMLType table named `purchaseorder`. Replace the `<TODO>` with the appropriate syntax to create the table. This script also loads data into the newly created table, and then gathers some statistics.
2. Run the script by using the XDBOE database connection.
3. Review the PURCHASEORDER table in SQL Developer. Use the Details tab in the right pane.

Practice 18-3: Using XQuery to Access XML Content

In this practice, you use XQuery to access XML content. XQuery is to XML content what SQL is to relational data. The XQuery standard was developed by the W3C and it is the natural language for querying, manipulating, and updating XML content. In this practice, you use XQuery to work with XML content stored in an Oracle database.

1. Use XQuery to count the number of documents in the PURCHASEORDER table by using the standard XQuery function `fn:collection()` to access the contents of the table. Remember that `fn:collection()` expects a path that identifies the set of XML documents to be processed. In this case, the path is prefixed with the “protocol” `oradb`, indicating that the components of the path should be interpreted as `DATABASE_SCHEMA` and `TABLE`.
2. Use XQuery to select a single document from the PURCHASEORDER table. Use a predicate on the `Reference` element to return one single document with the following `Reference` element: `AFRIPP-2012060818343243PDT`.
3. Use XQuery with multiple predicates, to return a single `Reference` node. The example shows you how to pass multiple predicate values into XQuery. Use `A60` as the predicate on `CostCenter`, `Diana Lorentz` as the `Requestor`, and `5` as the `Quantity`. Use the following expression as the last predicate: `count (LineItems/LineItem) > $QUANTITY] /Reference`
4. Review and run the `lab_18_03_04.sql` script. This script uses XQuery to construct a new summary document from the documents that match the specified predicates. This example also demonstrates the use of nested `FOR` loops, one for the set of `PurchaseOrder` documents, and another for the `LineItem` elements.
5. Use `XMLSerialize` in the previous query to convert the `XMLType` from the previous step to a `CLOB`. This allows the result to be viewed in SQL Developer versions that do not support rendering `XMLType`.
6. Review and run the `lab_18_03_06.sql` script, which demonstrates how you can use `XMLTable` to create an inline relational view from the documents that match the XQuery expression. This query shows you how to use the `columns` clause of the `XMLTable` operator to create an inline relational view from the documents that match the supplied predicates. In this case, the XQuery expression generates a result document from each of the `PurchaseOrder` documents that match the supplied predicates, and then the `columns` clause maps elements and attributes in the result document into the columns of the inline view. This allows a conventional relational result to be created by executing an XQuery operation on XML content.
7. Review and run the `lab_18_03_07.sql` script, which joins relational and XML tables by using XQuery.
8. You can use SQL Developer to examine the execution plans for XQuery expressions in the same way it can be used when working with SQL queries.
 - a. Using the Files tab, right-click the `sol_18_03_03.sql` script, and then select Open to display it in the SQL Worksheet area.
 - b. Position the cursor at the start of the Query in the `sol_18_03_03.sql` script, and then click the Autotrace icon. This displays the execution plan for the query.

Practice 18-4: Optimizing XQuery Performance with XML Indexing

Just like with SQL, XQuery operations can be improved by creating appropriate indexes. Again just like with SQL, XML indexing leads to trade-offs between DML and Query operations. In this practice, you use techniques for indexing binary XML content stored in the Oracle database.

1. Create a script to create a full unstructured XMLIndex on the PURCHASEORDER table named PURCHASEORDER_IDX. Next, add the following statement to the script to gather statistics: call dbms_stats.gather_schema_stats(USER).
2. Run the script as user XDBOE. This index will contain an entry for every node in every document. The full index requires no up-front knowledge about the structure of the XML being indexed. The index can be used to optimize both path (does the node exist?), and path-value (does the node exist and does it have a particular value?) searches.
3. Open the lab_18_04_03.sql script.
4. Select the XDBOE database connection from the Choose Db Connection drop-down list, position the cursor at the start of the first query, and click the Autotrace icon. Review the execution plan and determine whether XMLIndex is used.
5. Repeat step 4 for each of the remaining queries. In each case, the Autotrace output shows that XMLIndex is used to optimize the execution of the XQuery expressions.

Practice 18-5: Optimizing XQuery Performance by Using a Path-Subsetted XML Index

A full unstructured index can be an expensive proposition in both disk space usage and index maintenance overhead. To address these issues, XMLIndex provides two options:

- The first option is to maintain the index asynchronously. With an asynchronous index, the DML operation does not wait for indexing to complete before returning control to the application that invoked it. Instead, the indexing takes place in near real time, ensuring that index maintenance operations do not interfere with DML throughput.
- The second option is to use path-subsetting to explicitly include or exclude certain sections of an XML document from the index. This reduces both the size of the index and the overhead associated with index maintenance, by indexing only those nodes that will be referenced in predicates. While a path-subsetted index does not require upfront knowledge of the structure of the XML being indexed, it does require some knowledge of the kinds of searches that will be performed on the XML in question.

In this practice, you create a path-subsetted XMLIndex.

1. Create a path-subsetted XMLIndex on the PURCHASEORDER table. Narrow the focus of indexing by pruning the set of XPath expressions (paths) corresponding to XML fragments to be indexed, specifying a subset of all possible paths. This statement creates an index that indexes only top-level element PurchaseOrder and some of its children, as follows:
 - a. All Part elements and their descendants
 - b. All Reference elements
2. When gathering statistics for the optimizer on an XMLType table that is stored object-relationally, Oracle recommends that you gather statistics on all of the tables defined by the XML schema—that is, all of the tables in USER_XML_TABLES. You can use procedure DBMS_STATS.gather_schema_stats to do this, or use DBMS_STATS.gather_table_stats on each such table. This informs the optimizer about all of the dependent tables that are used to store the XMLType data. Use the DBMS_STATS.gather_schema_stats procedure to gather statistics.
3. Run the script as user XDBOE.
4. Return to the SQL worksheet containing the lab_18_04_03.sql script or re-open the lab_18_04_03.sql script.
5. Use Autotrace to generate the execution plan for each of the four queries. The Autotrace output shows that the XMLIndex is used when the XQuery expression contains nodes that are included in the index. However, when the XQuery expression does not contain any of the nodes in the index, the explain plans show that the optimizer reverts to using streaming XPath evaluation.

Practice 18-6: Optimizing XQuery Performance by Using a Structured XML Index

The unstructured index is best suited for working with highly unstructured XML. In situations where the XML is highly structured, but no XML schema is available, or the XML is semi-structured, containing islands of structure floating in a sea of unstructured content, the Structured XML Index can be used to optimize operations on the XML.

With a structured XML Index, one or more `XMLTable` operators are used to define which nodes in the document will be indexed. The selected nodes are projected into a series of relational tables that form the basis of the index. When the index has been created, the system identifies the XQuery expressions that can be evaluated by using the index, and uses the index appropriately. Secondary indexes can be created on the tables that make up the index, enabling further performance optimizations.

Structured indexes are very useful for optimizing XPath expressions that return a set of sibling nodes, or when the predicates in an XQuery expression reference sibling nodes. With an unstructured index, an independent index operation is required to find each of the nodes in question and then further processing is required to ensure that the selected nodes are indeed siblings. With a structured index, all of the nodes for a given XPath expression are stored in the same row and can be accessed in a single index operation, and no additional processing is required for sibling verification.

In this practice, you create a structured XML Index.

1. Open the `lab_18_06_01.sql` script. This script creates a structured XML Index. The definition of the XML Index is provided using `DBMS_XMLINDEX.createParameters`. Using a `parameters` clause simplifies the actual DDL needed to create the index. This is very useful when indexing complex XML structures, with multiple levels of nesting. Note how each `XMLTable` operator has a table name associated with it. The script also creates B-TREE indexes on the tables underlying the index to further optimize performance.
2. Run the script using the `XDBOE` database connection.
3. Return to the SQL worksheet containing the `lab_18_04_03.sql` script or re-open the `lab_18_04_03.sql` script. Use Autotrace to generate the execution plan for each of the four queries. The Autotrace output shows that the XML Index is used when the XQuery expression contains nodes that are included in the index. However, when the XQuery expression does not contain any of the nodes in the index, the explain plans show that the optimizer reverts to using streaming XPath evaluation.

Practice 18-7: Updating XML Content by Using XQuery-Update

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. Support for XQuery-Update is available in Oracle Database 11.2.0.3.0 and later. In Oracle XML DB XQuery operations are executed by using the XMLQuery Operator. This practice provides a brief introduction to the XQuery-Update standard and shows you how to use the XMLQuery () operator to perform XQuery-Update operations on XML content stored in the Oracle database. Though it appears that the result of an XQuery-Update operation is always a new document, in the background the XQuery-Update is re-written into a series of modifications to the data stored on disk. This enables partial update of the XML document, leading to significant savings in re-indexing and undo and redo generation.

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 the documents to which the XQuery-Update operation is applied. Predicates supplied to the XQuery-Update operation will determine which nodes within the documents selected by the XMLExists operation are actually updated.

In this practice, you use the XQuery-Update feature.

1. Open the lab_18_07_01.sql script. The query (INITIAL_STATE) generates a summary of the existing state of the document that is the target of the XQuery-Update operations. Replace <TODO> with the required XMLExists syntax, which uses a predicate on the Reference element to check if the value of the text node associated with the Reference element is 'AFRIPP-20120430212831873PDT' .
2. Run the script using the XDBOE database connection.
3. Open and run the lab_18_07_03.sql script. This XQuery-Update operation replaces the values of existing scalar nodes. Note how a predicate is used to determine which of the three description nodes in the document will be updated.
 - a. Replace the <TODO1> with the appropriate syntax to replace the User node with the value passed as USERID.
 - b. Replace the <TODO2> with the appropriate syntax to replace the Requestor node with the value passed as FULLNAME.
 - c. Replace the <TODO3> with the appropriate syntax to replace the Description attribute of LineItems/LineItem/Part that has a Description attribute value equals to the value passed as OLDTITLE with the value passed as NEWTITLE.
4. Open and run the lab_18_07_04.sql script. This XQuery operation (UPDATED_NODES) generates a summary showing the state of the document after the XQuery operation has been completed.

5. Open and run the `lab_18_07_05.sql` script using the XDBOE database connection. Replace the `<TODO>` with the appropriate syntax to delete the node where the `ItemNumber` attribute value of the `/PurchaseOrder/LineItems/LineItem` equals the value passed as `ITEMNO`. A predicate is used to determine which of the `LineItem` nodes in the document will be deleted.
 6. Write the query to confirm that the document from the previous step is deleted.
 7. Open, update, and then run `lab_18_07_07.sql` script using the XDBOE database connection. Use XQuery-Update to insert a new `LineItem` node into the referenced document. Use a predicate to determine where the new `LineItem` node will be placed in the document. Insert the new `LineItem` node after `/PurchaseOrder/LineItems/LineItem` where the `ItemNumber` attribute for the `LineItem` equals "3". Replace the `<TODO>` with the appropriate syntax.
 8. Open and run the `lab_18_07_08.sql` script by using the XDBOE database connection. The XQuery operation (`INSERTED_NODE`) shows that document has been updated. Since the new `LineItem` element (which has an `ItemNumber` attribute of 4) was placed after the `LineItem` element with an `ItemNumber` attribute of 3 the new `LineItem` is the 3rd `LineItem` element.
 9. Open and run the `lab_18_07_09.sql` script. Write using the XDBOE database connection. The XQuery-Update statement updates a fragment. It replaces the entire `LineItems` element with a new `LineItems` element. Revert the changes made to the `UserID` and `Requestor` elements.
- Note:** Because Oracle Database provides complete transaction control as part of the SQL language, the entire setup of updates could also have been undone by issuing a `rollback` command.
10. Write a query (`FINAL_STATE`) to confirm that the document is back in its original state.

Practice 18-8: Searching XML Content by Using XQuery Full Text

XQuery Full Text is an extension to W3C XQuery standard that makes it perform complex full-text-style search operations on the content of XML documents. Support for XQuery Full Text is available in Oracle Database 12.1.0.1.0 and later. The original XQuery language included an operator called `CONTAINS` that allowed pattern matching-based searches on XML content. The `CONTAINS` operator provides a case-sensitive substring-style search capability; that is, it returns `TRUE` if the source string contains exactly the set of characters contained in the target string. The XQuery Full Text specification adds the ability to perform word-based searches of XML content, with all of the common features of a text retrieval system. XQuery Full Text includes support for word match, windowing (word must appear within n words of word) and stemming (automatically recognize related words). In order to provide an efficient implementation of XQuery Full Text, Oracle Database 12c makes use of Oracle's text indexing technology. To use XQuery Full Text it is necessary to create a XML-aware full-text index on the documents that are to be searched.

In this practice, you create an XML Aware full-text index.

1. Create an XML Aware full-text index using the information in the following two tables. The first step is to define the section group and storage preferences that will be used by the index. These items are managed using methods provided by the package `CTX_DLL`. In order to use this package the `XBBOE` user must have been granted the role `CTXAPP`. The second step is to create a `CTXSYS.CONTEXT` index-based on the section group and storage preferences.

Procedure	Parameters
<code>CREATE_SECTION_GROUP</code>	<code>XQFT, PATH_SECTION_GROUP</code>
<code>SET_SEC_GRP_ATTR</code>	<code>XQFT, XML_ENABLE, TRUE</code>
<code>create_preference</code>	<code>STORAGE_PREFS, BASIC_STORAGE</code>
<code>set_attribute</code>	<code>STORAGE_PREFS, D_TABLE_CLAUSE, LOB(DOC) STORE AS SECUREFILE (COMPRESS MEDIUM CACHE)</code>
<code>set_attribute</code>	<code>STORAGE_PREFS, I_TABLE_CLAUSE, LOB(TOKEN_INFO) STORE AS SECUREFILE (NOCOMPRESS CACHE)</code>

Index Name	PURCHASEORDER_XQFT_IDX
Index Type	<code>CTXSYS.CONTEXT</code>
Index Parameters	<code>storage STORAGE_PREFS section group XQFT</code>

2. Open, examine, and run the `lab_18_08_02.sql` script by using the XDBOE database connection. This script performs XQuery Full Text searches on the XML content that is stored in Oracle XML DB.
3. Open, examine, and run the `lab_18_08_03.sql` script by using the XDBOE database connection. This query searches for an exact match on a phrase.
4. Open, examine, and run the `lab_18_08_04.sql` script using the XDBOE database connection. The query uses the XQuery `contains()` operator to search for an exact match on a phrase.
5. Open, examine, and run the `lab_18_08_05.sql` script by using the XDBOE database connection. This query uses the XQuery Full Text `contains` text operation instead of the `contains()` operator. The `contains` text operation searches for the target as a word in the source.
6. Open, examine, and run the `lab_18_08_06.sql` script by using the XDBOE database connection. This query uses the XQuery Full Text `using stemming` operation. The `using stemming` operation searches for any word related to the target as a word in the source.
7. Open, examine, and run the `lab_18_08_07.sql` script by using the XDBOE database connection. This query demonstrates how you can use XQuery Full Text to search fragments, as well as leaf-level nodes. In this query, the complex element `Address` is being searched for the word `Oxford`.
8. Open, examine, and run the `lab_18_08_08.sql` script by using the XDBOE database connection. This query demonstrates the use of the `ftand` operator to search for the presence of two words. The words do not need to be adjacent and do not need to be in any particular order.
9. Open, examine, and run the `lab_18_08_09.sql` script by using the XDBOE database connection. This query demonstrates how to add a window to the `ftand` operator. A window makes it possible to control how many words are allowed to exist between the two terms that are being searched on.
10. Open, examine, and run the `lab_18_08_10.sql` script using the XDBOE database connection. This query demonstrates how expanding the size of the window allows the search to return documents that were eliminated from the result set by the narrow window used in the previous query.

Practice 18-9: Optimizing XML Storage and Processing with XML Schema

XML schemas are used by many industries to define the XML structures used for information exchange. XML documents conforming to these XML schemas are highly structured. In this practice, you use an XML schema to optimize storage, index, query, and manage structured XML data.

Note: User credentials are case-sensitive. For example, for a username to be authenticated, it must match exactly the name as it was created. The default is all uppercase letters. In this example, you enter the username as XDBOE and the password as oracle.

Viewing the PurchaseOrder XML Schema Document Using a Web Browser

1. Launch your Firefox web browser. Click the browser icon.
2. Enter the following URL: <http://localhost:8080/home/XDBOE>
Content in the XML DB repository is protected by Access Control Lists (ACLs). In order to access the content, it is necessary to authenticate with the appropriate username and password. When the browser opens the Authentication Required dialog box, enter XDBOE for the username and oracle for the password, and then click OK. The browser displays the contents of the folder in question.
3. Click the `PurchaseOrder.xsd` document link to display the PurchaseOrder XML schema.

XML Schema Registration and Object-Relational Storage

Before an XML schema can be used with Oracle XML DB, it must be registered with the database. XML schemas can be registered for use with either binary XML storage or object-relational XML Storage. The decision on whether to use binary XML or object-relational storage will be driven by a number of factors, including:

- Complexity of the object-model defined by the XML schema
- Access patterns for instance documents
- Types of update operations that will take place
- Nature and frequency of changes to the XML schema

In general, if the XML schema describes a hierarchical object model with limited amounts of recursion and variability, then the XML schema is probably a candidate for object-relational storage. On the other hand, if the XML schema defines a complex, recursive structure with large degrees of variability and significant numbers of “any” elements, then binary XML storage is probably a better option.

Object-relational storage provides highly optimized leaf-level access and update, whereas binary XML storage is more efficient for document-level operations. Consequently in ETL scenarios, where the objective is to get SQL-based access to XML content or populate existing relational tables with values extracted from the XML, object-relational storage is probably more effective. Content-centric scenarios, where document-level operations are more common, are probably better suited for binary XML, because it avoids the overhead associated with a complete decomposition and reconstruction of the XML.

Another factor to consider is XML schema evolution. If the XML schema changes in ways that do not invalidate the existing corpus of instance documents, then object-relational storage's in-place schema evolution will probably be able to manage the schema changes. On the other hand, if the XML schema changes in ways that invalidate the existing corpus of documents on a regular basis, then schema-based binary XML or even non-schema-based binary XML is probably more appropriate.

Object-relational storage works by analyzing the object model defined by the XML schema and deriving an equivalent SQL object model. It then creates the set of tables that allow the SQL object model to be persisted in the database. Collections in the XML model are mapped to nested tables in the SQL model. A complete, lossless bi-directional mapping between the XML and SQL models ensures that XML documents are stored in the database with no loss of fidelity. The XML abstraction, based on `XMLType`, allows developers to manipulate XML by using XQuery. XQuery operations on object-relational `XMLType`, are compiled into the same query algebra as SQL, allowing the Oracle Database to optimize XQuery in the same way that it optimizes SQL. In this practice, you register an XML schema for use with object-relational storage and create indexes on the underlying storage model.

4. Open, examine, and run the `lab_18_09_04.sql` script by using the `XDBOE` database connection. This script annotates and registers the XML schema. It then describes some of the generated objects, and finally renames the nested tables used to manage the collections of `Action` and `LineItem` elements.

Bulk Loading XML Files by Using SQL Loader

Oracle's SQL Loader utility provides a convenient way of loading large volumes of XML documents into the database. The following example shows a SQL Loader control file that will load a set of documents into an XMLType table. SQL Loader can be used with binary and object-relational storage models, and works with both XMLType tables and XMLType columns.

```
options (direct=true)
load data
infile '2012.dat'
append
into table PURCHASEORDER
xmltype(XMLDATA) (
filename filler char(120),
XMLDATA lobfile(filename) terminated by eof)
```

In this example, direct-path loading is used to improve performance. The files will be loaded into the PURCHASEORDER table. The names of the files to be loaded are contained in the file 2012.dat. Each file name is a maximum of 120 characters long.

The file 2012.dat contains 10,000 entries in the following format.

```
sampleData/2012/Apr/ABANDA-20120407201438314PDT.xml
sampleData/2012/Apr/ABANDA-20120417140257543PDT.xml
sampleData/2012/Apr/ABANDA-20120427125050959PDT.xml
. . .
```

In this practice, you use SQL Loader to load the sample documents into the database.

Note: There is no solution file for this practice.

1. Close SQL Developer.
2. Open a command-line terminal window.
3. Execute the following command: cd \$XDB_HOL/sqlLoader

The sqlLoader folder contains the 2012.ctl and 2012.dat files that you will need in this practice.

Note: In the setup section of this case study, you set the environment variable as follows: export XDB_HOL="/home/oracle/Desktop/DatabaseTrack/XMLDB"

4. Execute the following command so that SQL Loader will direct path load 10,000 documents into the PURCHASEORDER table:
sqlldr -userid=XDBOE/oracle -control=2012.ctl -rows=1000
5. Once SQL Loader completes successfully, close the terminal window.

Practice 18-10: Using XQuery with Object-Relational Storage

One of the key features of Oracle XML DB is that applications are developed independently of the way in which the XML is stored. This means that the choice of storage model does not affect the way in which applications are written. In the previous practice, the binary XML version of the PURCHASEORDER table was replaced with one based on object-relational storage.

In this practice, you observe how no changes are required to execute the XQuery expressions used with the binary XML table on the object-relational table.

1. Start SQL Developer. Open the `lab_18_10_01.sql` script by using the XDBOE database connection.
Note: This script contains the same individual queries that you created and ran in Practice 18-03.
2. Click the Run Script icon to execute the script. The output is displayed on the Script Output tab.
3. Open the `lab_18_10_03.sql` script.
4. Use Autotrace to generate the execution plan for each of the queries. The Autotrace output shows that full table scans are being used to execute the queries. This is expected because, currently, there are no indexes that can be used to optimize the queries.

Highlight query 1, and then click the Autotrace icon on the SQL Worksheet toolbar. The output is displayed on the Autotrace tab. Repeat this step for the remaining queries.

Practice 18-11: Indexing Object-Relational Storage to Optimize XQuery Operations

XQuery operations on object-relational storage are optimized by creating conventional B-Tree indexes on the tables that are used to persist the SQL objects. There are two ways of determining which tables and columns should be indexed.

The first involves generating the execution plan for the query and then creating indexes based on the information in the plan. In this case, the index will be created by using the system-generated table and column names shown in the execution plan.

The second involves using the method `XPATH2TABCOLMAPPING` in the package `DBMS_XMLSTORAGE_MANAGE` to determine which table and column correspond to a particular XPath expression, and then using this information to generate a DDL statement that creates the appropriate index.

In this practice, you use the second approach to create indexes to optimize queries on the `Reference` and `User` nodes and a compound index on the repeating elements `Part Number` and `Quantity`.

1. Open the `lab_18_11_01.sql` script.
2. The script uses the `DBMS_XMLSTORAGE_MANAGE` package to determine which tables and columns need to be indexed. The output from running these queries is a set of DDL statements that will create the required indexes. Click the Run Script icon to execute the script. The output is displayed in the Script Output tab.
3. Click the **Run Script Output as Script** icon. This opens a new SQL Worksheet that contains the create index statements.
4. Position the cursor at the beginning of the first “create index” statement, and then click the Run Statement icon. This creates the index needed to optimize queries on the `Reference` element.
5. Repeat step 4 for each of the remaining “create index” statements as well as to call the `DBMS_STATS.GATHER_SCHEMA_STATS` package procedure.
6. Return to the SQL Worksheet containing the `lab_18_10_03.sql` script or re-open the `lab_18_10_03.sql` script. Use Autotrace to generate the execution plan for each of the four queries. The Autotrace output now shows that the `REFERENCE_INDEX`, `USER_INDEX`, and `PART_NUMBER_QUANTITY_INDEX` indexes are used to optimize the XQuery operations.
7. Highlight query 1 or place the cursor at the start of the query, and then click the Autotrace icon on the SQL Worksheet tab. Repeat this step for the remaining queries in this script.

Practice 18-12: Equi-Partitioning of Object-Relational Storage

Partitioning helps simplify XML data lifecycle management and performance. Equi-partitioning for object-relational storage was introduced in Oracle Database 11g Release 2. Equi-partitioning ensures that all of the nested tables used to manage collections automatically follow the partitioning strategy defined for the parent XMLType table. This allows partitioning pruning and partition maintenance operations to operate on XMLType tables in the same way as they operate on simple relational tables.

1. Open and run the `lab_18_12_01.sql` script by using the XDBOE database connection.
2. This script uses a Create Table As Select (CTAS) operation to make a copy of the content of the existing PURCHASEORDER table. It then drops the PURCHASEORDER table and re-creates it as a partitioned table. The table is partitioned based on the value of the User element. Note that the partition key has to be specified using object-relational syntax. The script then reloads the documents into the PURCHASEORDER table and re-creates the indexes. `REFERENCE_INDEX` is created as a global index, while indexes `USER_INDEX` and `PART_NUMBER_QUANTITY_INDEX` are created as local indexes.
3. Open the `lab_18_10_01.sql` script.
4. Use Autotrace to generate the execution plan for each of the seven queries. The Autotrace output shows that the table is now partitioned, parallel execution occurs where appropriate, and partition pruning takes place for XQuery operations that include a predicate on the partition key.

Practice 18-13: Relational Access to XML Content

Oracle's implementation of the SQL language provides many powerful features that are not yet supported by the XQuery standard. Also there are many tools (and developers) that are not yet able to work effectively with XML content and XQuery. One of the key benefits of XML DB is that it provides relational tools and programmers with the ability to use SQL to work directly with XML content, enabling all the features of SQL to be used with XML and preserving existing investments in relational technology and infrastructure.

1. Open the `lab_18_13_01.sql` script.
2. Run the script using the `XDBOE` database connection.

This script uses XQuery and `XMLTable` to create two views that provide relational access to XML content. These views provide a typical MASTER-DETAIL relationship that can be used to access the contents of the XML using SQL. The views use XQuery to map scalar values from the XML into the columns defined by the view.

The first view provides access to values from elements and attributes that occur at most once per document. It can be considered the master table.

The second view provides access to values from the collection of `LineItem` elements that can occur multiple times per document. It can be considered the detail table. This view is created using nested `XMLTable` operators.

The first `XMLTable` operator projects out the values that form the primary key for each document and the set of repeating elements. The set of repeating elements forms the input to the second `XMLTable` operator, which generates one row for each of the repeating elements.

Because the second `XMLTable` processes the output of the first `XMLTable`, a correlated join takes place. This ensures that the rows produced by the secondary `XMLTable` are joined with the corresponding rows from the primary `XMLTable`.

Examine the Views in SQL Developer

1. In this section, you examine the views that you created in the previous step, which you can use with any tool that supports accessing relational data via SQL views. Select the Connections tab, and then click the + icon to the left of the `XDBOE` connection to expand it.
2. Expand Views, and then select `PURCHASEORDER_MASTER_VIEW`.
3. In the right pane, click the Columns tab, if not already selected.
Note: SQL Developer considers the view a standard relational view.
4. Click the Data tab.
Note: SQL Developer is able to retrieve the data, even though the content is stored in the database as an `XMLType` table.

5. Repeat steps 2, 3, and 4 for the PURCHASEORDER_DETAIL_VIEW.
This demonstrates that any ID, reporting tool, or Business Intelligence tool that understands the relational paradigm and SQL can use this technique to access content stored in XMLType tables within Oracle Database.
Expand Views in the XDBOE connection, and then select PURCHASEORDER_DETAIL_VIEW.
6. In the right pane, click the Columns tab, if not already selected. Note that SQL Developer sees the view as a standard relational view.
7. Click the Data tab. Note again that SQL Developer is able to retrieve the data, even though the content is stored in the database as an XMLType table.

Performing SQL Operations on XML Content

8. In this section, you demonstrate how relational views of XML enable simple and advanced SQL statements to operate on XML content stored in the Oracle database. Open the lab_18_13_08.sql script.
9. Run the script using the XDBOE database connection. None of the queries contain a reference to XML constructs. Advanced SQL functionality like group by, group by (rollup()), and lag can be used to analyze XML content. Predicates can be supplied as part of the SQL queries. The output is displayed in the Script Output tab.
10. Use Autotrace to generate the execution plan for each of the queries. The execution plan shows that all of the queries are executed as SQL queries over the tables that manage the XML content. The XML abstraction has been completely bypassed in these cases.

Practice 18-14: XML Access to Relational Content

The previous practice demonstrated how to access XML content from SQL. Oracle XML DB also allows XQuery to operate directly on relational data. Two distinct metaphors 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 on 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.

Using XQuery to Generate XML Documents From Relational Tables

1. Open the `lab_18_14_01.sql` script. The first example uses XQuery and `fn:collection` to generate XML documents from relational tables.
2. Run the script by using the XDBOE database connection.

The first query shows the output of the simple canonical mapping used to generate XML documents from rows in a relational table. 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; character escaping will be used when the column name contains characters that are not permitted as an XML name. XQuery pragmas can be used to determine how null values are handled.

The second query shows how XQuery can be used to transform the canonical mapping into a more useful XML format. This example also shows how this technique can be used to generate documents containing content that comes from multiple relational tables.

3. Use Autotrace to generate the execution plan for the second query.

The Autotrace output shows that a purely relational execution plan is used to generate the XML. As expected, the execution plan for the SQL/XML approach is just as efficient as the execution plan for the XQuery based option.

Using SQL/XML Functions to Generate XML Documents from Relational Tables

4. The second example in this section demonstrates how to use SQL/XML functions to generate XML documents from relational tables. Open the `lab_18_14_04.sql` script.
5. Run the script using the XDBOE database connection.

This query shows the SQL/XML equivalent of the second XQuery in the previous example. SQL/XML uses a template-based approach to generating XML from relational data. The nesting of the SQL/XML operators describes the shape of the document and how the column values are mapped into text nodes and attributes.

6. Use Autotrace to generate the execution plan for the second query.

The Autotrace output shows that a purely relational execution plan is used to generate the XML. As expected the execution plan for the SQL/XML approach is just as efficient as the execution plan for the XQuery-based option.

Practice 18-15: Creating a Persistent XML View of Relational Data

In this practice, you create an XML view of relational data. You use the same SQL/XML query as you did in the previous example; however the XQuery expression from the first example in this practice could also be used for this purpose. The view is a view of `XMLType`, which means that the view definition must include a mechanism for generating a unique ID for each row in the view. In this example the ID is supplied by the expression in the `with object id` clause of the create view statement. In this case, uniqueness is based on the contents of the `Name` element.

1. Open the `lab_18_15_01.sql` script.
2. Run this script by using the `XDBOE` database connection.
3. Open the `lab_18_15_03.sql` script.
4. Run the script by using the `XDBOE` database connection. The `XMLType` view allows XQuery operations to be performed against relational data. In the same way that relational views of XML enable SQL-centric development on top of XML content, XML views of relational data enable XML-centric developers to work with relational data.
5. Use Autotrace to generate the execution plan for the second query. The Autotrace output shows that the execution plan obtained when executing an XQuery on a `XMLType` view of relational data is a purely relational plan, demonstrating that XML DB is able to optimize XQuery operations on relational data in the same way that it can optimize relational operations on XML content.

Summary

After completing the case study, you should have learned how to:

- Review the steps and scripts that are required to set up the environment for the case study
- Create a new database connection for the user for this case study, and check the Autotrace parameters
- Access XML content by using XQuery
- Use XMLIndex, path-subsetted XMLIndex, and structured XMLIndex to optimize XQuery performance
- Use XQuery-Update to update XML content
- Create an XML-aware full-text index and perform XQuery Full Text searches on XML content
- Optimize XML storage with XML schema



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

Summary

After completing the case study, you should have learned how to:

- Use the same XQuery queries that are used with binary XML storage with object-relational storage
- Index object-relational storage to optimize XQuery operations
- Equi-partition object-relational storage
- Create relational views from XML content
- Use SQL/XML functions to generate XML documents from relational tables
- Create a persistent XML view of relational 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

A

Table Descriptions

ORACLE®

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

Schema Descriptions

Overall Description

The Oracle Database sample schemas portray a sample company that operates worldwide to fill orders for several different products. The company has two divisions:

- **Human Resources (HR):** Tracks information about the employees and facilities
- **Order Entry (OE):** Tracks product inventories and sales through various channels

Each of these divisions is represented by a schema. In this course, you have access to the objects in all the schemas. However, the emphasis of the examples, demonstrations, and practices is on the HR and OE schemas.

All scripts necessary to create the sample schemas reside in the \$ORACLE_HOME/demo/schema/ folder.

Human Resources (HR)

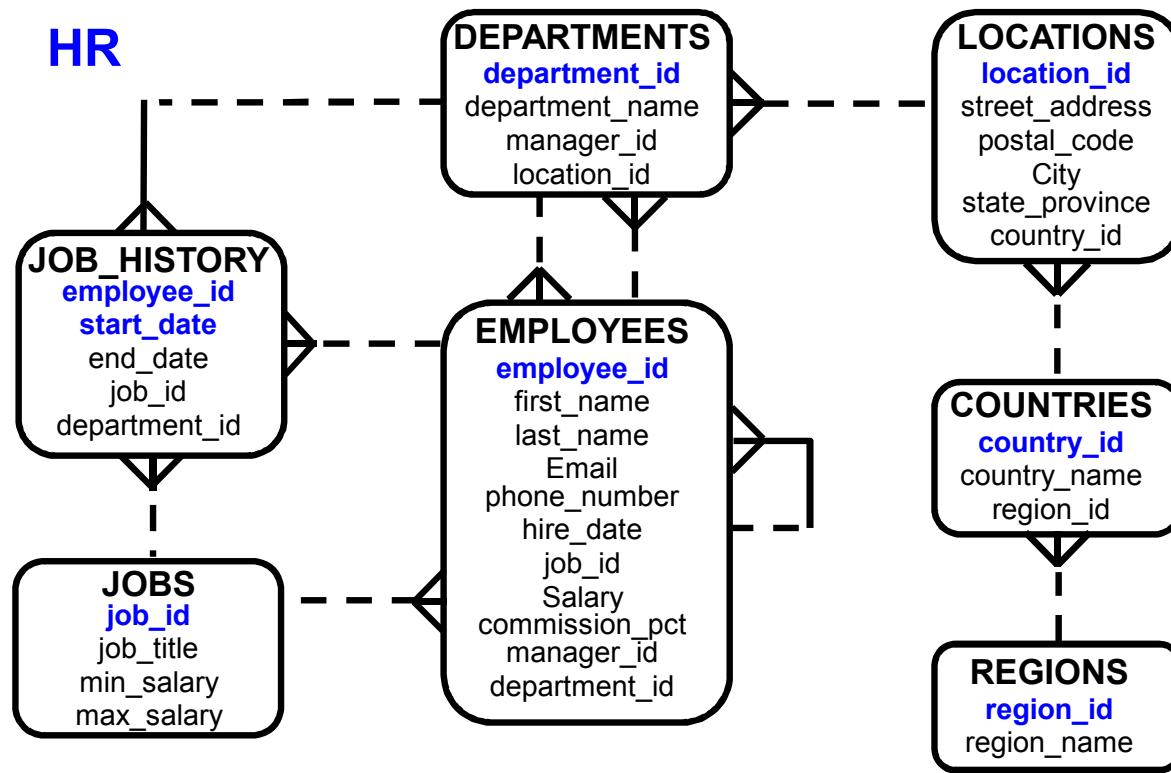
In the HR records, each employee has an identification number, email address, job identification code, salary, and manager. Some employees earn commissions in addition to their salary.

The company also tracks information about the 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 that the employee was 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 geographical region where the country is located.

The HR Entity Relationship Diagram



Human Resources (HR) Row Counts

```
SELECT COUNT(*) FROM regions;
```

```
  COUNT(*)
```

```
-----  
        4
```

```
SELECT COUNT(*) FROM countries;
```

```
  COUNT(*)
```

```
-----  
        25
```

```
SELECT COUNT(*) FROM locations;
```

```
  COUNT(*)
```

```
-----  
        23
```

```
SELECT COUNT(*) FROM departments;
```

```
  COUNT(*)
```

```
-----  
        27
```

```
SELECT COUNT(*) FROM jobs;
```

```
  COUNT(*)
```

```
-----  
        19
```

```
SELECT COUNT(*) FROM employees;
```

```
  COUNT(*)
```

```
-----  
        107
```

```
SELECT COUNT(*) FROM job_history;
```

```
  COUNT(*)
```

```
-----  
        10
```

Order Entry (OE)

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 the product will be sold, and a URL for manufacturer information.

Inventory information is also recorded for all the products, including the warehouse where the product is available and the quantity on hand. Because the products are sold worldwide, the company maintains the names of the products and their descriptions in different languages.

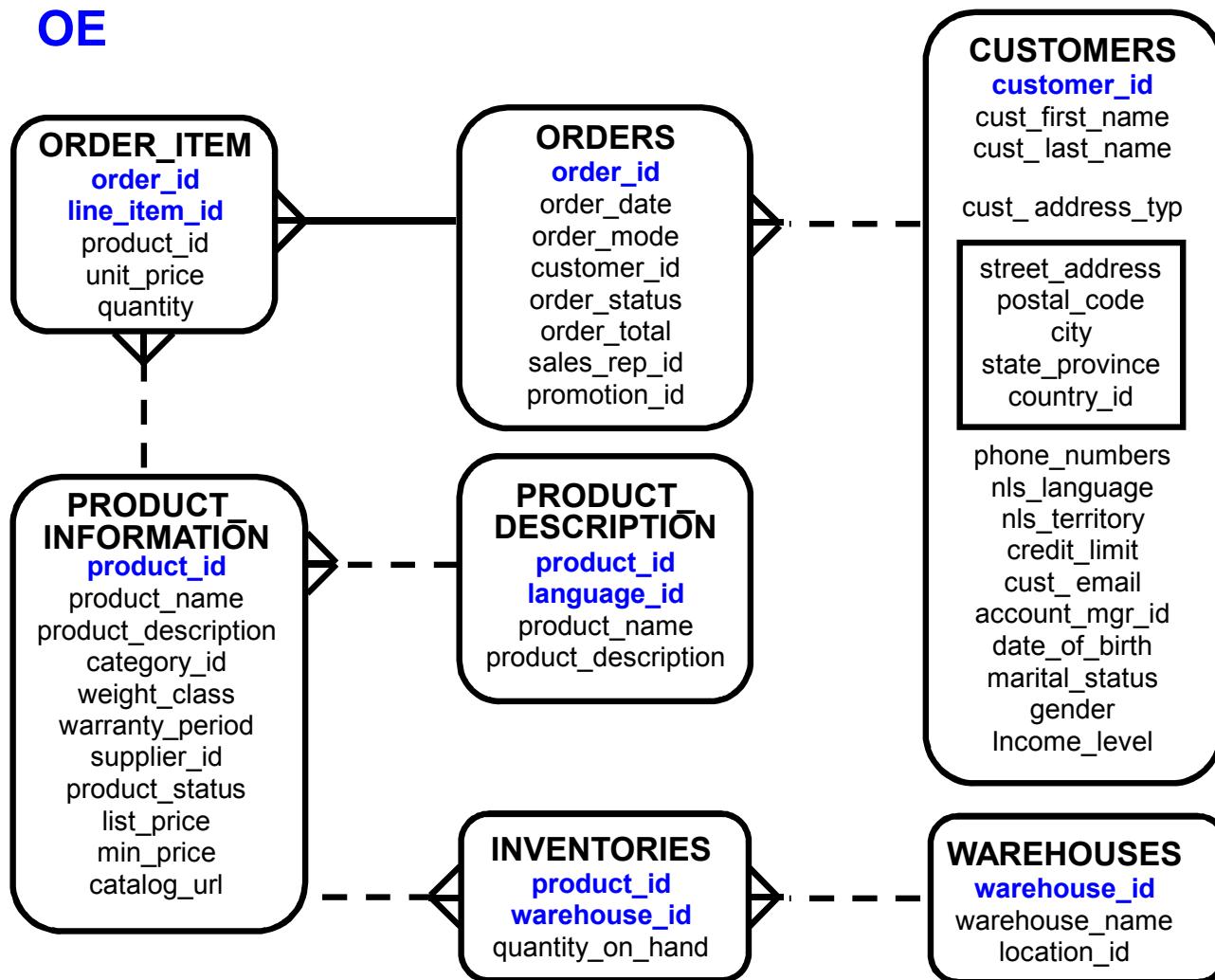
The company maintains warehouses in several locations to facilitate filling customer orders. Each warehouse has a warehouse identification number, name, and location identification number.

Customer information is tracked in some detail. Each customer is assigned an identification number. Customer records include name, street address, city or province, country, phone numbers (up to five phone numbers for each customer), and postal code. Some customers order through the Internet, so email addresses are also recorded. Because of language differences among the customers, the company records the NLS language and territory of each customer. The company places a credit limit on its customers to limit the amount for which they can purchase at one time. Some customers have account managers, whom the company monitors. A customer's phone number is also tracked.

When a customer places an order, the company tracks the date of the order, the mode of the order, status, shipping mode, total amount of the order, and the sales representative who helped place the order. This may or may not be the same individual as the account manager for a customer, or in the case of an order over the Internet, the sales representative is not recorded. In addition to the order information, the number of items ordered, the unit price, and the products ordered are also tracked.

For each country in which it does business, the company records the country name, currency symbol, currency name, and the geographical region where the country is located. This data is useful to interact with customers living in different geographical regions around the world.

Order Entry (OE)



Order Entry (OE) Row Counts

```
SELECT COUNT(*) FROM customers;
```

```
  COUNT(*)
```

```
-----
```

```
 319
```

```
SELECT COUNT(*) FROM inventories;
```

```
  COUNT(*)
```

```
-----
```

```
 1112
```

```
SELECT COUNT(*) FROM orders;
```

```
  COUNT(*)
```

```
-----
```

```
 105
```

```
SELECT COUNT(*) FROM order_items;
```

```
  COUNT(*)
```

```
-----
```

```
 665
```

```
SELECT COUNT(*) FROM product_descriptions;
```

```
  COUNT(*)
```

```
-----
```

```
 8640
```

```
SELECT COUNT(*) FROM product_information;
```

```
  COUNT(*)
```

```
-----
```

```
 288
```

```
SELECT COUNT(*) FROM warehouses;
```

```
  COUNT(*)
```

```
-----
```

```
 9
```

Using SQL Developer

ORACLE

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

Objectives

After completing this appendix, you should be able to:

- List the key features of Oracle SQL Developer
- Identify the menu items of Oracle SQL Developer
- Create a database connection
- Manage database objects
- Use SQL Worksheet
- Save and run SQL scripts
- Create and save reports
- Browse the Data Modeling options in SQL Developer

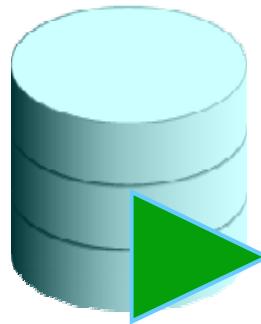


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In this appendix, you are introduced to the graphical tool called SQL Developer. You learn how to use SQL Developer for your database development tasks. You learn how to use SQL Worksheet to execute SQL statements and SQL scripts.

What Is Oracle SQL Developer?

- Oracle SQL Developer is a 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.



SQL Developer

ORACLE

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

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

SQL Developer, which is 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.

SQL Developer is the interface to administer the Oracle Application Express Listener. The new interface enables you to specify global settings and multiple database settings with different database connections for the Application Express Listener. SQL Developer provides the option to drag and drop objects by table or column name onto the worksheet. It provides improved DB Diff comparison options, GRANT statements support in the SQL editor, and DB Doc reporting. Additionally, SQL Developer includes support for Oracle Database 12c features.

Specifications of SQL Developer

- Is shipped along with Oracle Database 12c Release 1
- Is developed in Java
- Supports Windows, Linux, and Mac OS X platforms
- Enables default connectivity using the JDBC Thin driver
- Connects to Oracle Database version 9.2.0.1 and later



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

Oracle SQL Developer is a free integrated development environment that simplifies the development and management of Oracle Database. SQL Developer offers complete end-to-end development of your PL/SQL applications, a worksheet for running queries and scripts, a DBA console for managing the database, a reports interface, a complete data modeling solution, and a migration platform for moving your 3rd party databases to Oracle.

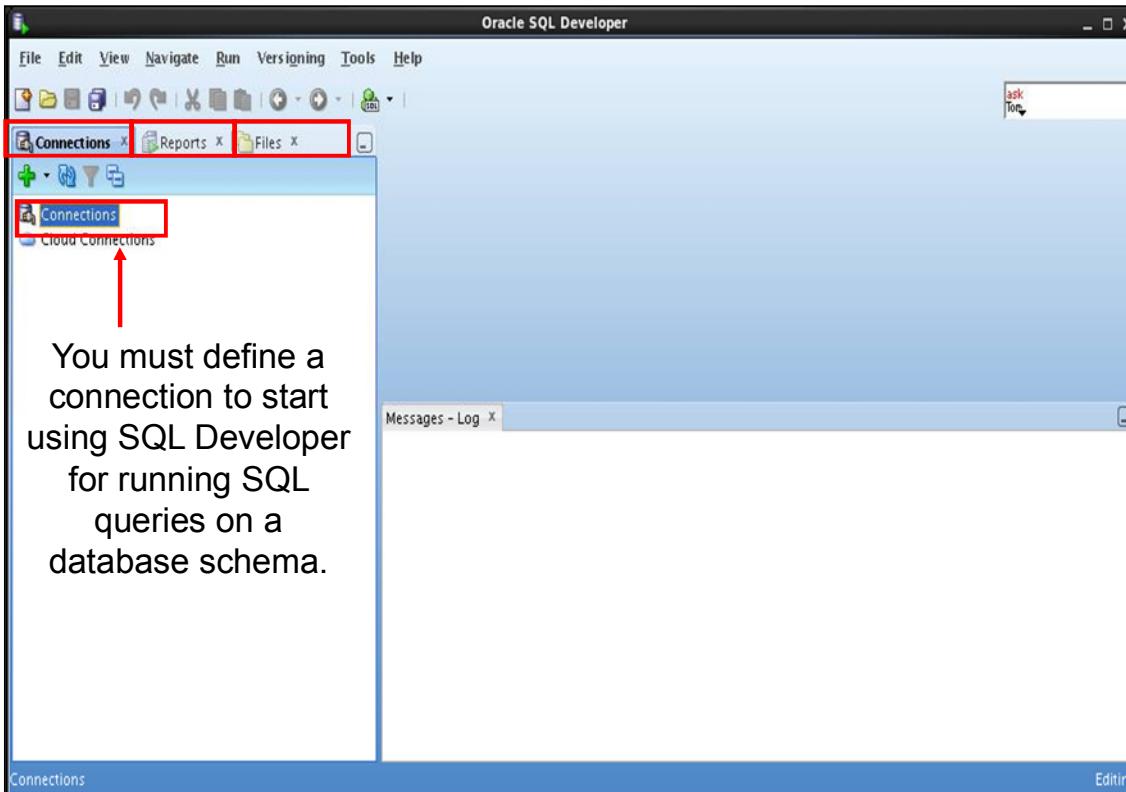
For Oracle Database 12c Release 1, you will have to download and install SQL Developer. SQL Developer is freely downloadable from the following link:

<http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html>

For instructions on how to install SQL Developer, see the following link:

<http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html>

SQL Developer 3.2 Interface



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

ORACLE

The SQL Developer interface contains three main navigation tabs, from left to right:

- **Connections tab:** By using this tab, you can browse database objects and users to which you have access.
- **Reports tab:** Identified by the Reports icon, this tab enables you to run predefined reports or create and add your own reports.
- **Files tab:** Identified by the Files folder icon, this tab enables you to access files from your local machine without having to use the File > Open menu.

General Navigation and Use

SQL Developer uses the left side for navigation to find and select objects, and the right side to display information about selected objects. You can customize many aspects of the appearance and behavior of SQL Developer by setting preferences.

Note: You need to define at least one connection to be able to connect to a database schema and issue SQL queries or run procedures and functions.

Menus

The following menus contain standard entries, plus entries for features that are specific to SQL Developer:

- **View:** Contains options that affect what is displayed in the SQL Developer interface
- **Navigate:** Contains options for navigating to panes and for executing subprograms
- **Run:** Contains the Run File and Execution Profile options that are relevant when a function or procedure is selected, and also debugging options
- **Versioning:** Provides integrated support for the following versioning and source control systems – Concurrent Versions System (CVS) and Subversion
- **Tools:** Invokes SQL Developer tools such as SQL*Plus, Preferences, and SQL Worksheet. It also contains options related to migrating third-party databases to Oracle.

Note: The Run menu also contains options that are relevant when a function or procedure is selected for debugging.

Creating a Database Connection

- You must have at least one database connection to use SQL Developer.
- You can create and test connections for:
 - Multiple databases
 - Multiple schemas
- SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.
- You can export connections to an Extensible Markup Language (XML) file.
- Each additional database connection created is listed in the Connections Navigator hierarchy.



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

A connection is a SQL Developer object that specifies the necessary information for connecting to a specific database as a specific user of that database. To use SQL Developer, you must have at least one database connection, which may be existing, created, or imported.

You can create and test connections for multiple databases and for multiple schemas.

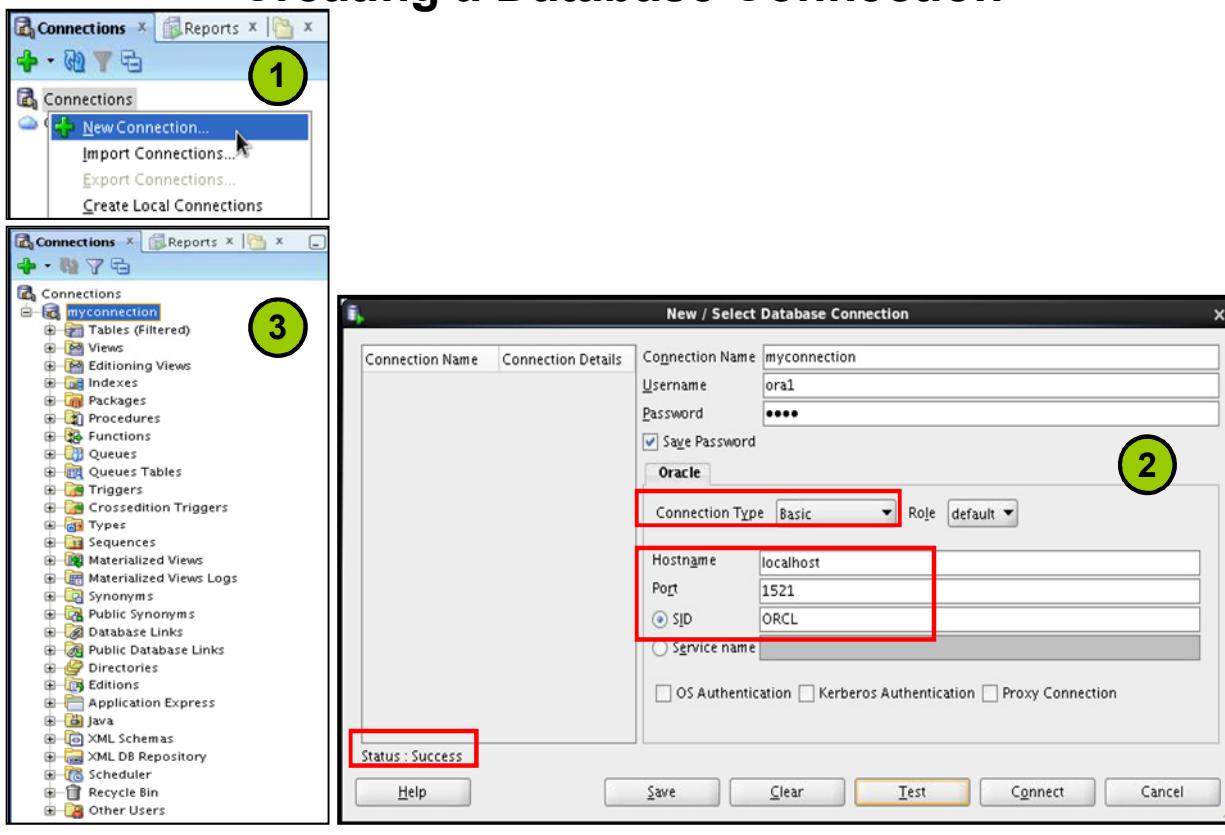
By default, the `tnsnames.ora` file is located in the `$ORACLE_HOME/network/admin` directory, but it can also be in the directory specified by the `TNS_ADMIN` environment variable or registry value. When you start SQL Developer and open the Database Connections dialog box, SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.

Note: On Windows, if the `tnsnames.ora` file exists, but its connections are not being used by SQL Developer, define `TNS_ADMIN` as a system environment variable.

You can export connections to an XML file so that you can reuse it.

You can create additional connections as different users to the same database or to connect to the different databases.

Creating a Database Connection



ORACLE

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

To create a database connection, perform the following steps:

1. On the Connections tabbed page, right-click Connections and select New Connection.
2. In the New/Select Database Connection window, enter the connection name. Enter the username and password of the schema that you want to connect to.
 - a. From the Role drop-down list, you can select either *default* or *SYSDBA*. (You choose *SYSDBA* for the *sys* user or any user with database administrator privileges.)
 - b. You can select the connection type as:
 - Basic:** In this type, enter host name and SID for the database that you want to connect to. Port is already set to 1521. You can also choose to enter the Service name directly if you use a remote database connection.
 - TNS:** You can select any one of the database aliases imported from the *tnsnames.ora* file.
 - LDAP:** You can look up database services in Oracle Internet Directory, which is a component of Oracle Identity Management.
 - Advanced:** You can define a custom Java Database Connectivity (JDBC) URL to connect to the database.

Local/Bequeath: If the client and database exist on the same computer, a client connection can be passed directly to a dedicated server process without going through the listener.

- c. Click Test to ensure that the connection has been set correctly.
- d. Click Connect.

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

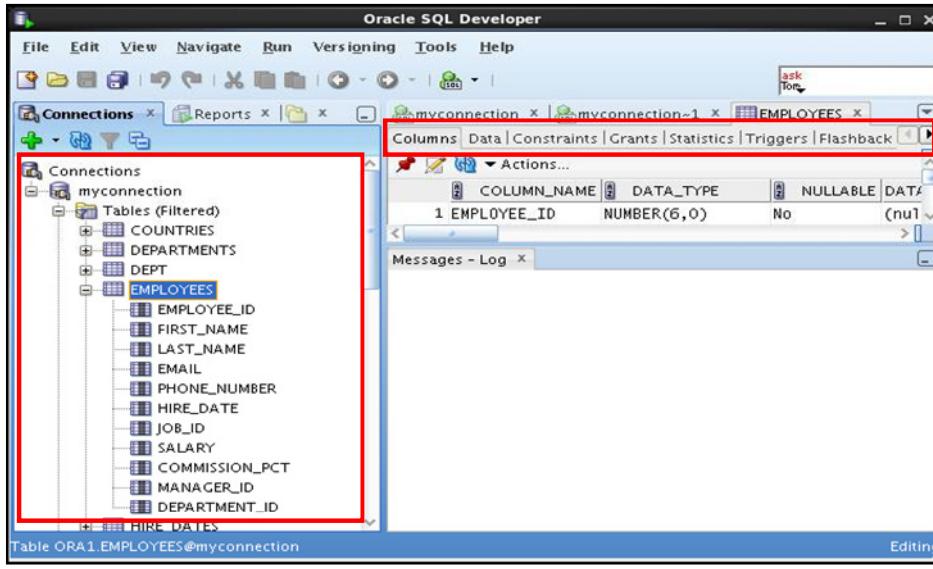
3. The connection gets added in the Connections Navigator. You can expand the connection to view the database objects and view object definitions(dependencies, details, statistics, and so on).

Note: From the same New>Select Database Connection window, you can define connections to non-Oracle data sources using the Access, MySQL, and SQL Server tabs. However, these connections are read-only connections that enable you to browse objects and data in that data source.

Browsing Database Objects

Use the Connections Navigator to:

- Browse through many objects in a database schema
- Review the definitions of objects at a glance



ORACLE

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

After you create a database connection, you can use the Connections Navigator to browse through many objects in a database schema, including Tables, Views, Indexes, Packages, Procedures, Triggers, and Types.

SQL Developer uses the left side for navigation to find and select objects, and the right side to display information about the selected objects. You can customize many aspects of the appearance of SQL Developer by setting preferences.

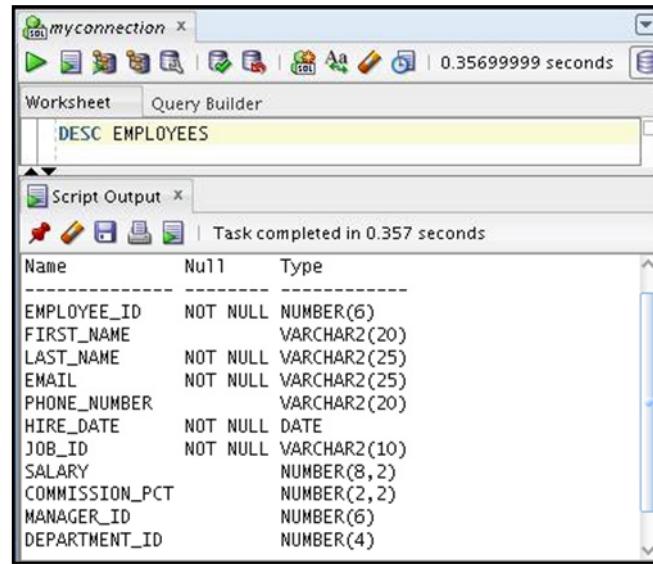
You can see the definition of the objects broken into tabs of information that is pulled out of the data dictionary. For example, if you select a table in the Navigator, details about columns, constraints, grants, statistics, triggers, and so on are displayed on an easy-to-read tabbed page.

If you want to see the definition of the EMPLOYEES table as shown in the slide, perform the following steps:

1. Expand the Connections node in the Connections Navigator.
2. Expand Tables.
3. Click EMPLOYEES. By default, the Columns tab is selected. It shows the column description of the table. Using the Data tab, you can view the table data and also enter new rows, update data, and commit these changes to the database.

Displaying the Table Structure

Use the DESCRIBE command to display the structure of a table:



The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there's a connection named "myconnection". Below it, the "Worksheet" tab is active, showing the command "DESC EMPLOYEES" in the text area. To the right of the worksheet is a "Script Output" tab which displays the results of the query. The results are presented as a table with three columns: Name, Null, and Type. The data shows the structure of the EMPLOYEES table, including columns like EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID, and DEPARTMENT_ID, along with their respective data types (NUMBER or VARCHAR2).

Name	Null	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

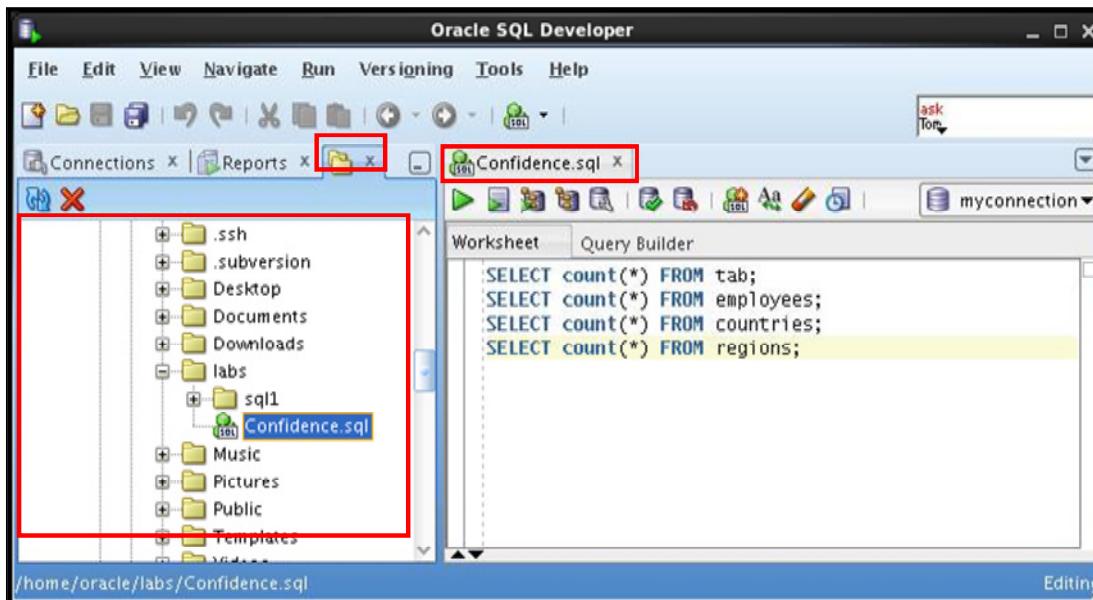
ORACLE

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

In SQL Developer, you can also display the structure of a table using the DESCRIBE command. The result of the command is a display of column names and data types, as well as an indication of whether a column must contain data.

Browsing Files

Use the File Navigator to explore the file system and open system files.



ORACLE

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

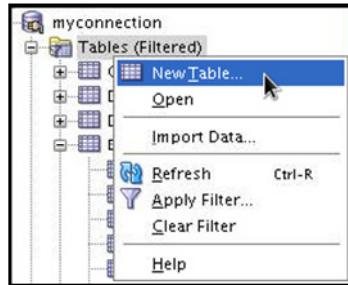
Browsing Database Objects

You can use the File Navigator to browse and open system files.

- To view the File Navigator, click the View tab and select Files, or select View > Files.
- To view the contents of a file, double-click a file name to display its contents in the SQL Worksheet area.

Creating a Schema Object

- SQL Developer supports the creation of any schema object by:
 - Executing a SQL statement in 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.



ORACLE

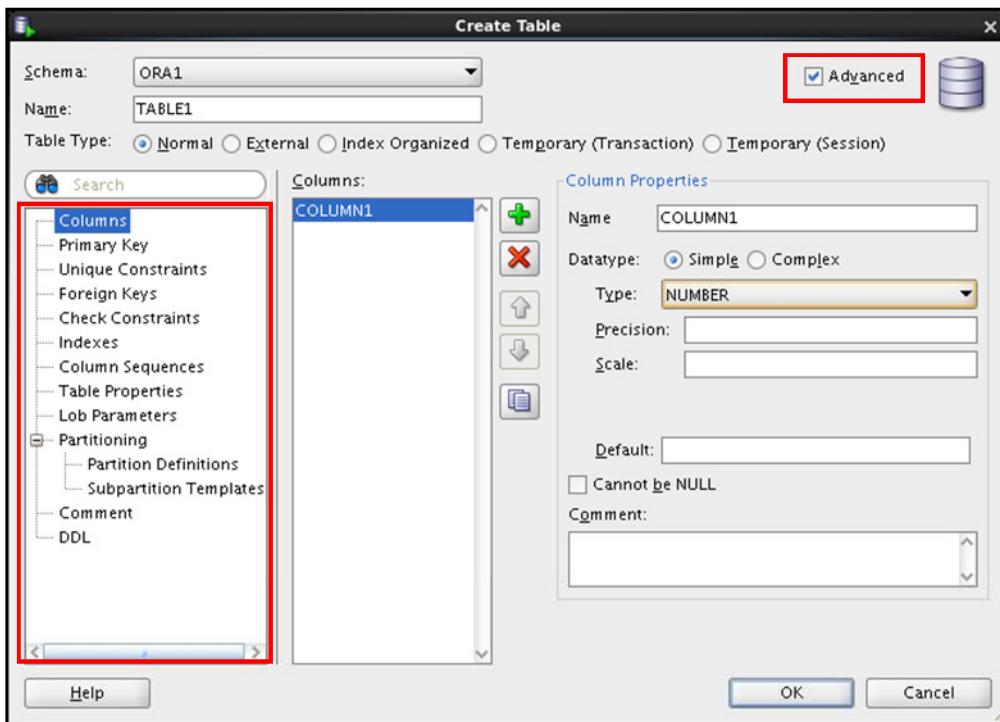
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

SQL Developer supports the creation of any schema object by executing a SQL statement in SQL Worksheet. Alternatively, you can create objects by using the context menus. When created, you can edit objects using an edit dialog box or one of the many context-sensitive menus.

As new objects are created or existing objects are edited, the DDL for those adjustments is available for review. An Export DDL option is available if you want to create the full DDL for one or more objects in the schema.

The slide shows how to create a table using the context menu. To open a dialog box for creating a new table, right-click Tables and select New Table. The dialog boxes to create and edit database objects have multiple tabs, each reflecting a logical grouping of properties for that type of object.

Creating a New Table: Example



ORACLE

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

In the Create Table dialog box, if you do not select the Advanced check box, you can create a table quickly by specifying columns and some frequently used features.

If you select the Advanced check box, the Create Table dialog box changes to one with multiple options, in which you can specify an extended set of features while you create the table.

The example in the slide shows how to create the DEPENDENTS table by selecting the Advanced check box.

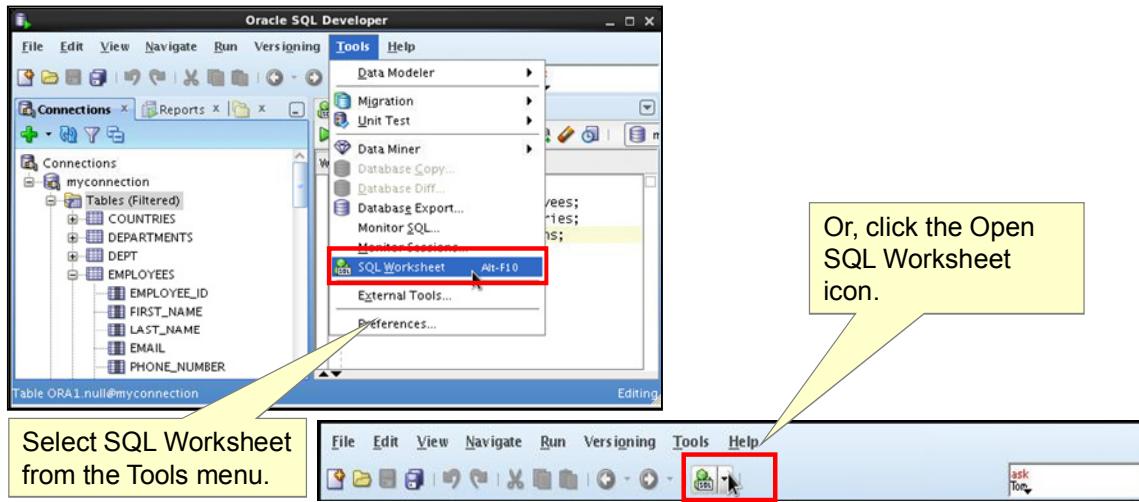
To create a new table, perform the following steps:

1. In the Connections Navigator, right-click Tables and select Create TABLE.
2. In the Create Table dialog box, select Advanced.
3. Specify the column information.
4. Click OK.

Although it is not required, you should also specify a primary key by using the Primary Key tab in the dialog box. Sometimes, you may want to edit the table that you have created; to do so, right-click the table in the Connections Navigator and select Edit.

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.



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

ORACLE

When you connect to a database, a SQL Worksheet window for that connection automatically opens. 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.

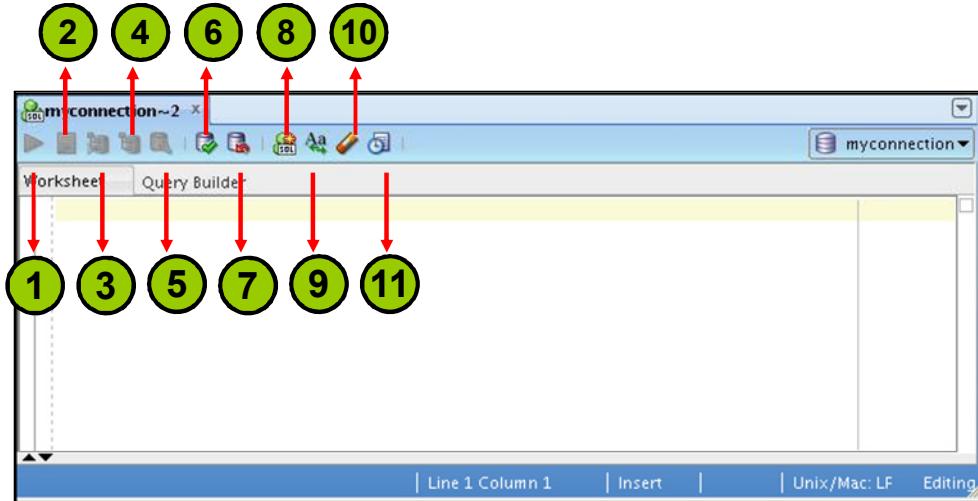
You can specify the actions that can be processed by the database connection associated with the worksheet, such as:

- 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 using one of the following:

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

Using the SQL Worksheet



The Oracle logo, consisting of the word "ORACLE" in white capital letters inside a red horizontal bar.

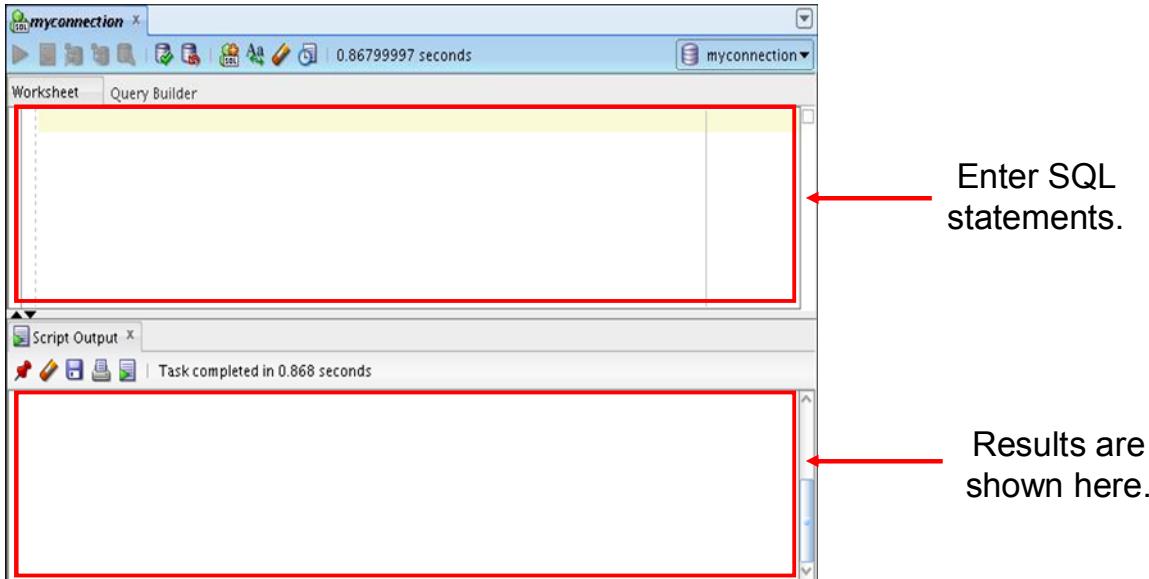
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

1. **Run Statement:** Executes the statement where the cursor is located in the Enter SQL Statement box. You can use bind variables in the SQL statements, but not substitution variables.
2. **Run Script:** Executes all the statements in the Enter SQL Statement box by using the Script Runner. You can use substitution variables in the SQL statements, but not bind variables.
3. **Autotrace:** Generates trace information for the statement
4. **Explain Plan:** Generates the execution plan, which you can see by clicking the Explain tab
5. **SQL Tuning Advisory:** Analyzes high-volume SQL statements and offers tuning recommendations
6. **Commit:** Writes any changes to the database and ends the transaction
7. **Rollback:** Discards any changes to the database, without writing them to the database, and ends the transaction

8. **Unshared SQL Worksheet:** Creates a separate unshared SQL Worksheet for a connection
9. **To Upper/Lower/InitCap:** Changes the selected text to uppercase, lowercase, or initcap, respectively
10. **Clear:** Erases the statement or statements in the Enter SQL Statement box
11. **SQL History:** Displays a dialog box with information about the SQL statements that you have executed

Using the SQL 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 automatically opens. You can use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. All SQL and PL/SQL commands are supported as they are passed directly from the SQL Worksheet to the Oracle database. The SQL*Plus commands that are used in SQL Developer must be interpreted by the SQL Worksheet before being passed to the database.

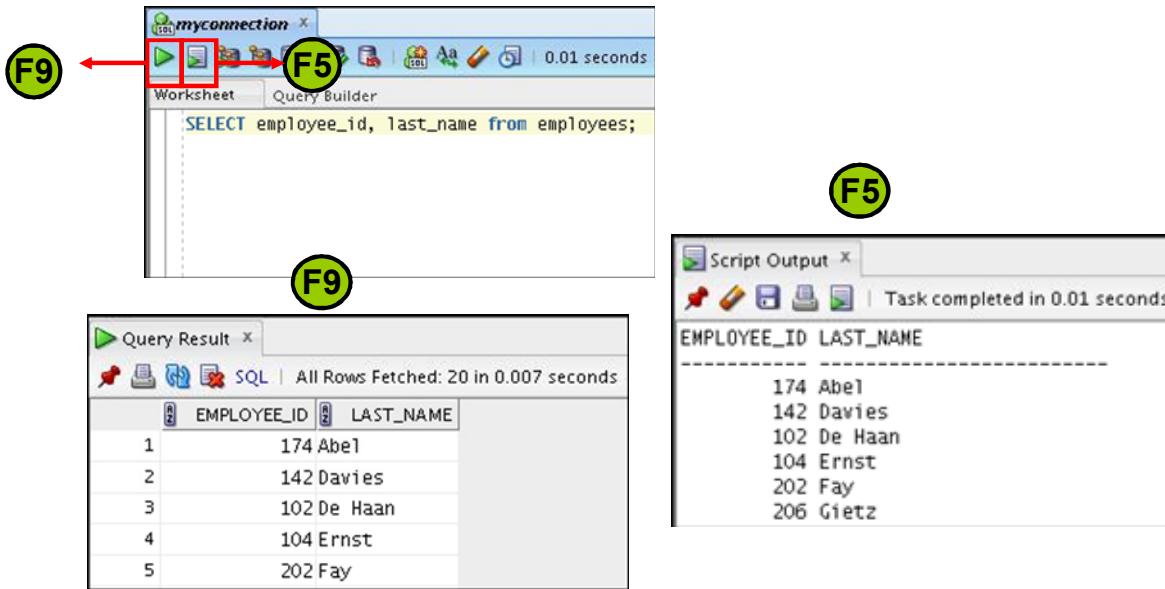
The SQL Worksheet currently supports a number of SQL*Plus commands. Commands that are not supported by the SQL Worksheet are ignored and not sent to the Oracle database. Through the SQL Worksheet, you can execute the SQL statements and some of the SQL*Plus commands.

You can display a SQL Worksheet by using any of the following options:

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

Executing SQL Statements

Use the Enter SQL Statement box to enter single or multiple SQL statements.

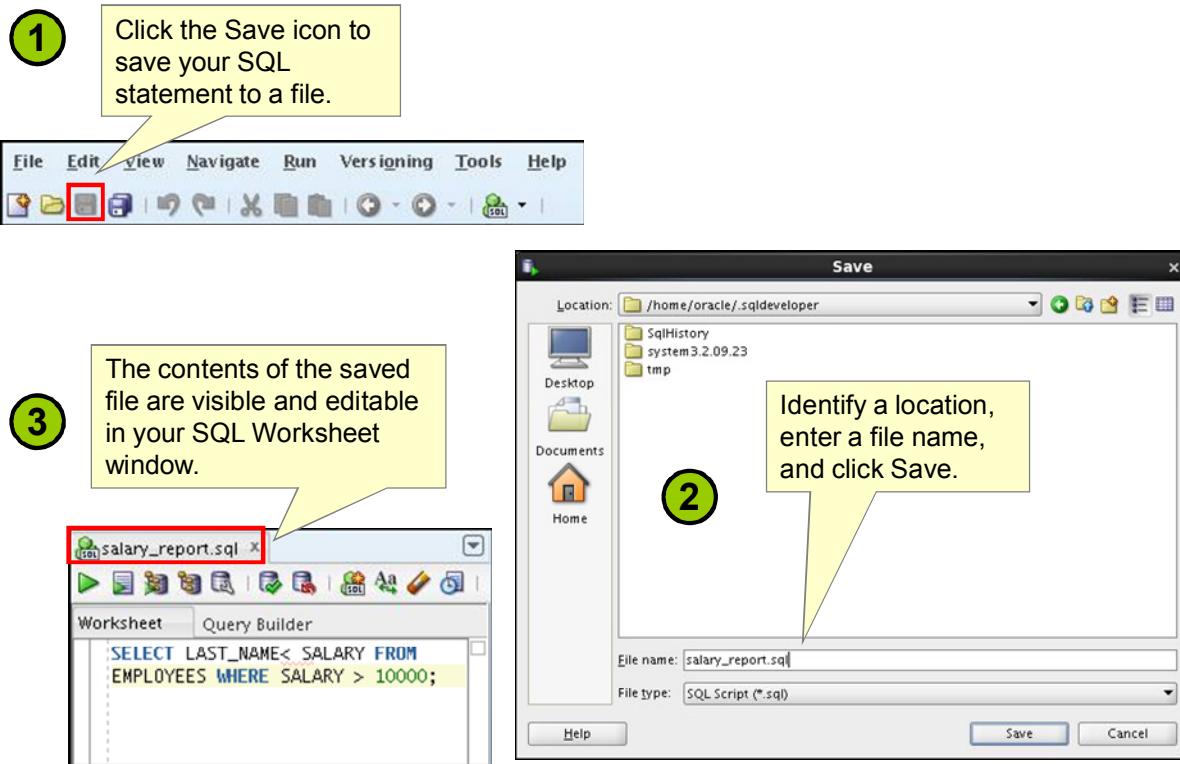


ORACLE

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

The example in the slide shows the difference in output for the same query when the F9 key or Execute Statement is used versus the output 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 to a text file. To save the contents of the Enter SQL Statement box, perform the following steps:

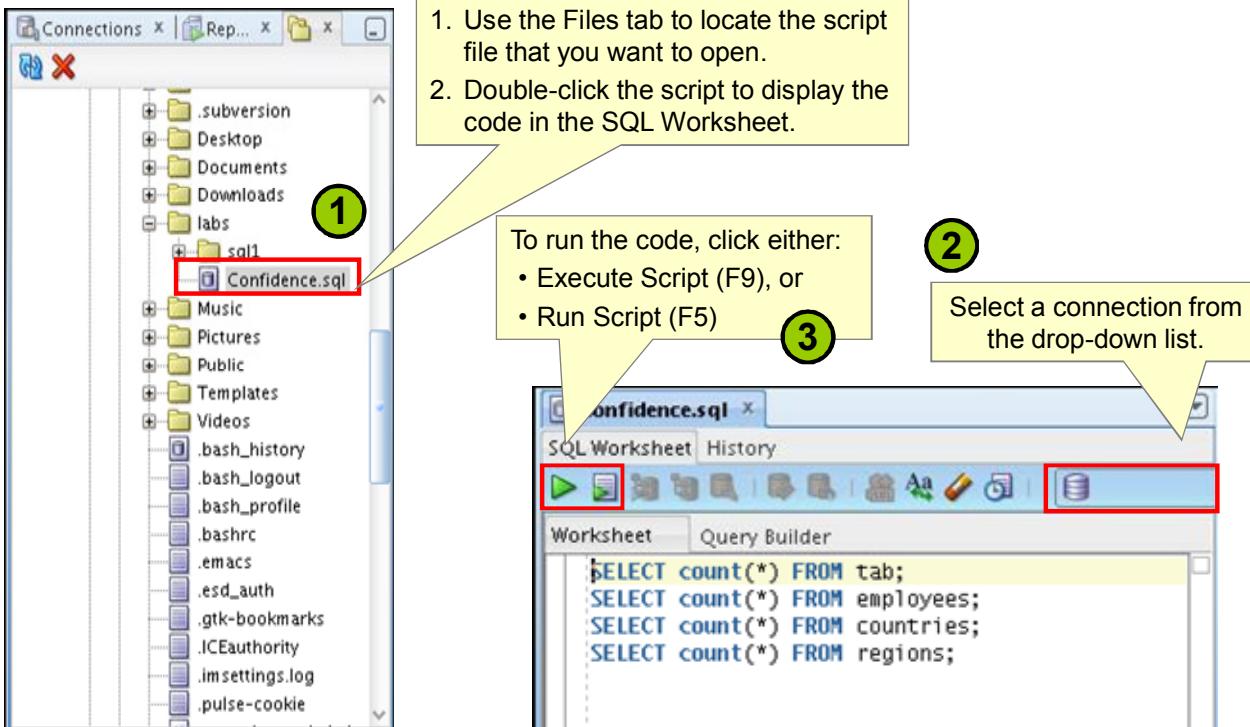
1. Click the Save icon or use the File > Save menu item.
2. In the Save dialog box, enter a file name and the location where you want the file saved.
3. Click Save.

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 displays 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: Method 1



ORACLE

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

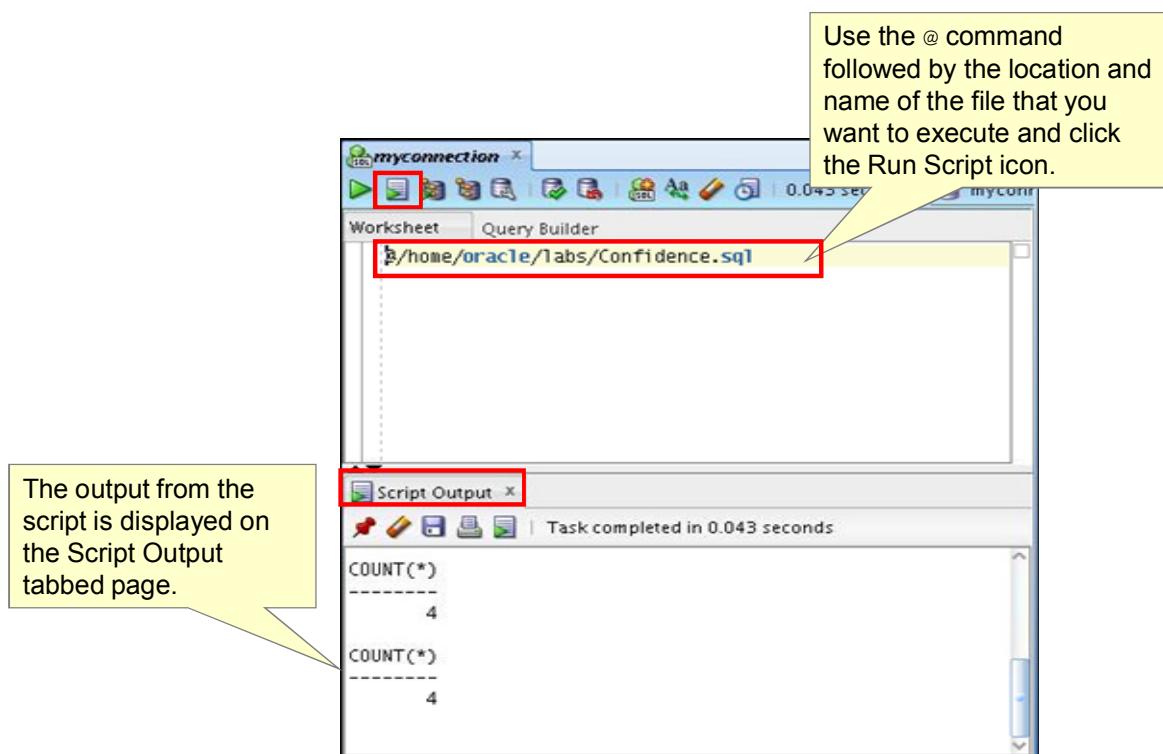
To open a script file and display the code in the SQL Worksheet area, perform the following steps:

1. In the files navigator, select (or navigate to) the script file that you want to open.
2. Double-click the file to open it. The code of the script file is displayed in the SQL Worksheet area.
3. Select a connection from the connection drop-down list.
4. To run the code, click the Run Script (F5) icon on the SQL Worksheet toolbar. If you have not selected a connection from the connection drop-down list, a connection dialog box will appear. Select the connection that you want to use for the script execution.

Alternatively, you can also do the following:

1. Select File > Open. The Open dialog box is displayed.
2. In the Open dialog box, select (or navigate to) the script file that you want to open.
3. Click Open. The code of the script file is displayed in the SQL Worksheet area.
4. Select a connection from the connection drop-down list.
5. To run the code, click the Run Script (F5) icon on the SQL Worksheet toolbar. If you have not selected a connection from the connection drop-down list, a connection dialog box will appear. Select the connection that you want to use for the script execution.

Executing Saved Script Files: Method 2



ORACLE

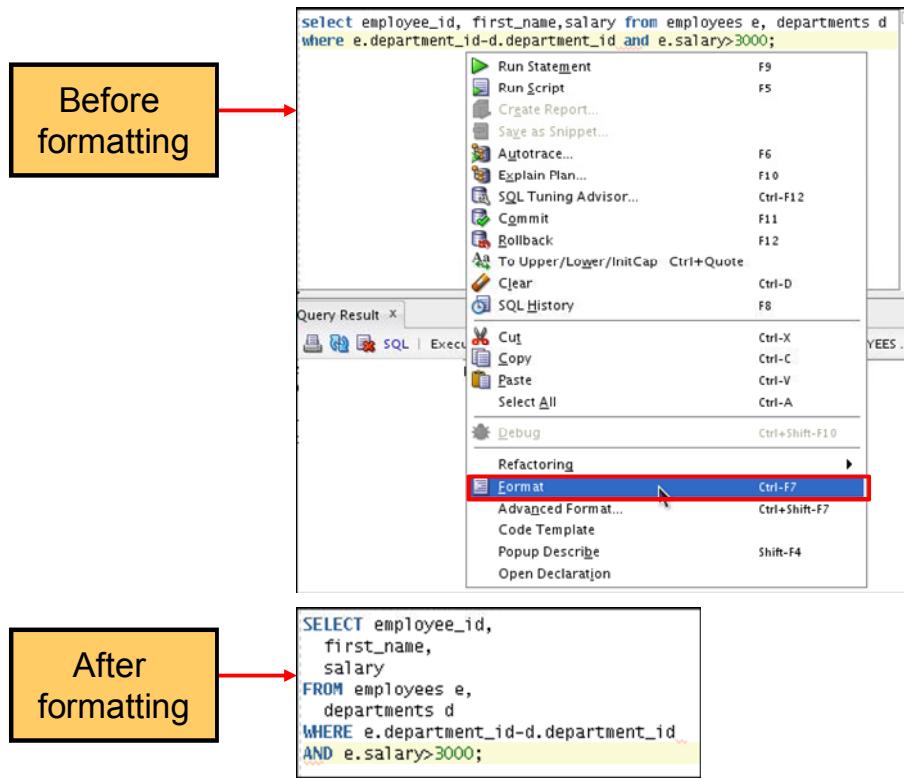
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To run a saved SQL script, perform the following steps:

1. Use the @ command followed by the location and the name of the file that you want to run in the Enter SQL Statement window.
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 File Save dialog box appears and you can identify a name and location for your file.

Formatting the SQL Code



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

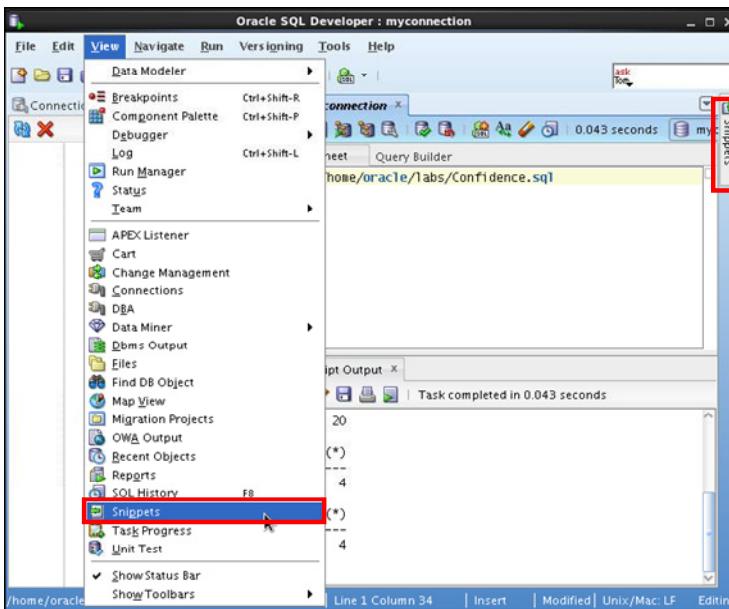
You may want to format the indentation, spacing, capitalization, and line separation of the SQL code. SQL Developer has a feature for formatting SQL code.

To format the SQL code, right-click in the statement area and select Format.

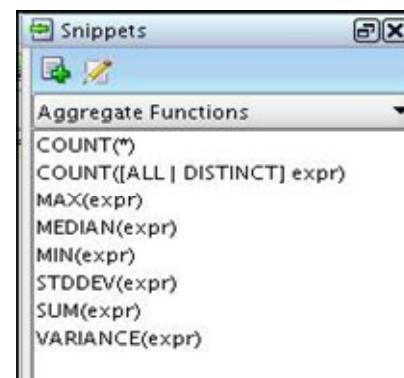
In the example in the slide, before formatting, the SQL code has the keywords not capitalized and the statement not properly indented. After formatting, the SQL code is beautified with the keywords capitalized and the statement properly indented.

Using Snippets

Snippets are code fragments that may be just syntax or examples.



When you place your cursor here, it shows the Snippets window. From the drop-down list, you can select the functions category that you want.



ORACLE

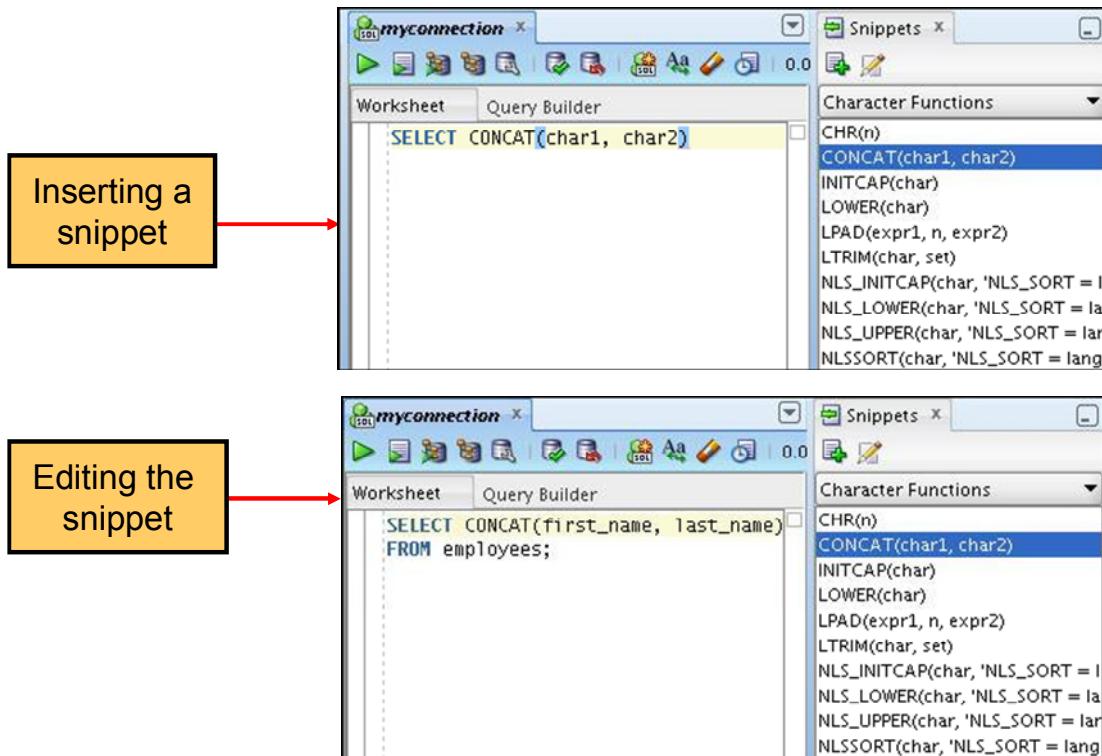
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You may want to use certain code fragments when you use the SQL Worksheet or create or edit a PL/SQL function or procedure. SQL Developer has a feature called Snippets. Snippets are code fragments such as SQL functions, optimizer hints, and miscellaneous PL/SQL programming techniques. You can drag snippets to the Editor window.

To display Snippets, select View > Snippets.

The Snippets window is displayed on the right. You can use the drop-down list to select a group. A Snippets button is placed in the right window margin, so that you can display the Snippets window if it becomes hidden.

Using Snippets: Example



ORACLE

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

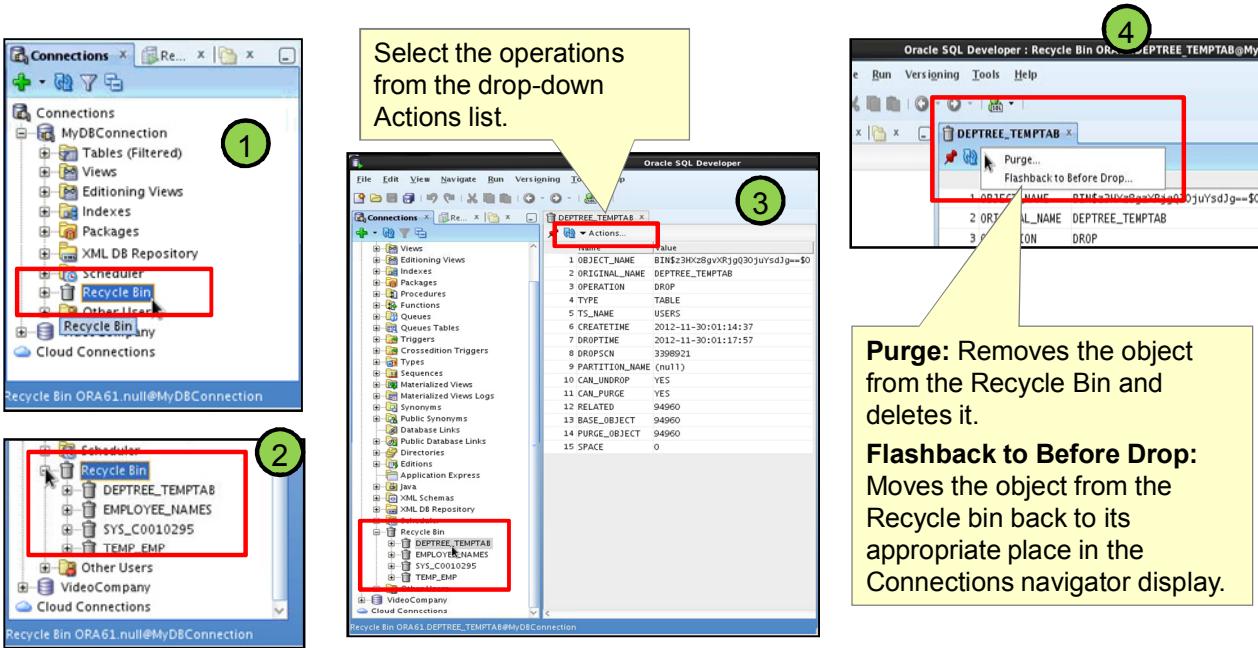
To insert a Snippet into your code in a SQL Worksheet or in a PL/SQL function or procedure, drag the snippet from the Snippets window to the desired place in your code. Then you can edit the syntax so that the SQL function is valid in the current context. To see a brief description of a SQL function in a tool tip, place the cursor over the function name.

The example in the slide shows that `CONCAT (char1, char2)` is dragged from the Character Functions group in the Snippets window. Then the `CONCAT` function syntax is edited and the rest of the statement is added as in the following:

```
SELECT CONCAT(first_name, last_name)  
FROM employees;
```

Using Recycle Bin

The Recycle Bin holds objects that have been dropped.



ORACLE

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

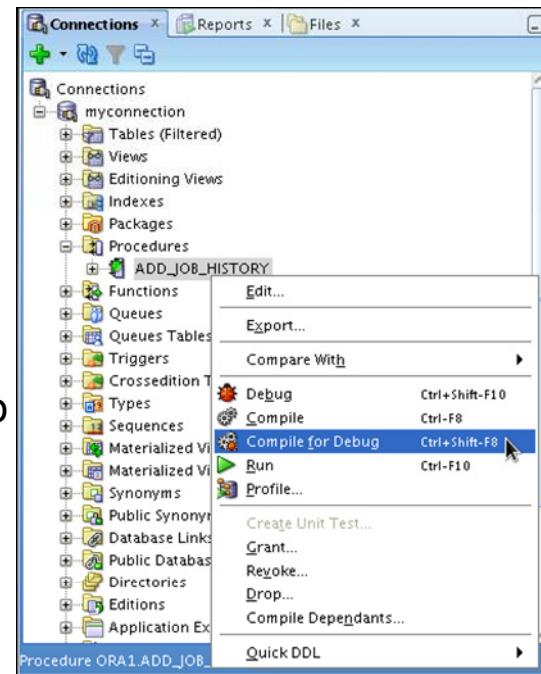
The recycle bin is a data dictionary table containing information about dropped objects. Dropped tables and any associated objects such as indexes, constraints, nested tables, and the likes are not removed and still occupy space. They continue to count against user space quotas, until specifically purged from the recycle bin or the unlikely situation where they must be purged by the database because of tablespace space constraints.

To use the Recycle Bin, perform the following steps:

1. In the Connections navigator, select (or navigate to) the Recycle Bin.
2. Expand Recycle Bin and click the object name. The object details are displayed in the SQL Worksheet area.
3. Click the Actions drop-down list and select the operation you want to perform on the object.

Debugging Procedures and Functions

- Use SQL Developer to debug PL/SQL functions and procedures.
- Use the Compile for Debug option to perform a PL/SQL compilation so that the procedure can be debugged.
- Use the Debug menu options to set breakpoints, and to perform step into, step over tasks.



ORACLE

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

In SQL Developer, you can debug PL/SQL procedures and functions. Using the Debug menu options, you can perform the following debugging tasks:

- **Find Execution Point** goes to the next execution point.
- **Resume** continues execution.
- **Step Over** bypasses the next method and goes to the next statement after the method.
- **Step Into** goes to the first statement in the next method.
- **Step Out** leaves the current method and goes to the next statement.
- **Step to End of Method** goes to the last statement of the current method.
- **Pause** halts execution, but does not exit, thus allowing you to resume execution.
- **Terminate** halts and exits the execution. You cannot resume execution from this point; instead, to start running or debugging from the beginning of the function or procedure, click the Run or Debug icon on the Source tab toolbar.
- **Garbage Collection** removes invalid objects from the cache in favor of more frequently accessed and more valid objects.

These options are also available as icons on the Debugging tab of the output window.

Database Reporting

SQL Developer provides a number of predefined reports about the database and its objects.

Owner	Name	Type	Referenced_Owner	Referenced_Name	Referenced_Type
APEX_040100	APEX	PROCEDURE	APEX_040100	WW_FLOW	PACKAGE
APEX_040100	APEX	PROCEDURE	APEX_040100	WW_FLOW_ISC	PACKAGE
APEX_040100	APEX	PROCEDURE	APEX_040100	WW_FLOW_SECURITY	PACKAGE
APEX_040100	APEX	PROCEDURE	SYS	STANDARD	PACKAGE
APEX_040100	APEX	PROCEDURE	SYS	SYS_STUB_FOR_PURITY_ANALYSIS	PACKAGE
APEX_040100	APEXWS	PACKAGE	SYS	STANDARD	PACKAGE
APEX_040100	APEXADMIN	PROCEDURE	APEX_040100	F	PROCEDURE
APEX_040100	APEX_ADMIN	PROCEDURE	SYS	STANDARD	PACKAGE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	NV	FUNCTION
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOWS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_APPLICATION_GROUPS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_AUTHENTICATIONS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_COMPANIES	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_COMPANY_SCHEMAS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_COMPUTATIONS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_ICON_BAR	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_INSTALL_SCRIPTS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_ITEMS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_LANGUAGE_MAP	TABLE

ORACLE

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

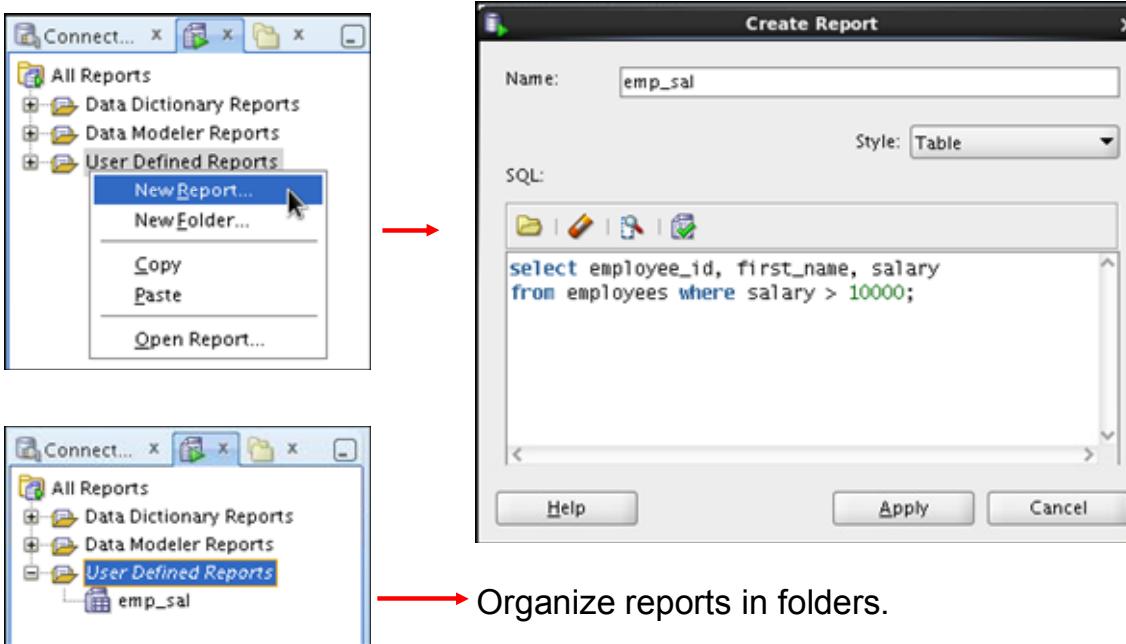
SQL Developer provides many reports about the database and its objects. These reports can be grouped into the following categories:

- About Your Database reports
- Database Administration reports
- Table reports
- PL/SQL reports
- Security reports
- XML reports
- Jobs reports
- Streams reports
- All Objects reports
- Data Dictionary reports
- User-Defined reports

To display reports, click the Reports tab on the left of the window. Individual reports are displayed in tabbed panes on the right of the window; for each report, you can select (using a drop-down list) the database connection for which to display the report. For reports about objects, the objects shown are only those visible to the database user associated with the selected database connection, and the rows are usually ordered by Owner. You can also create your own user-defined reports.

Creating a User-Defined Report

Create and save user-defined reports for repeated use.



ORACLE

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

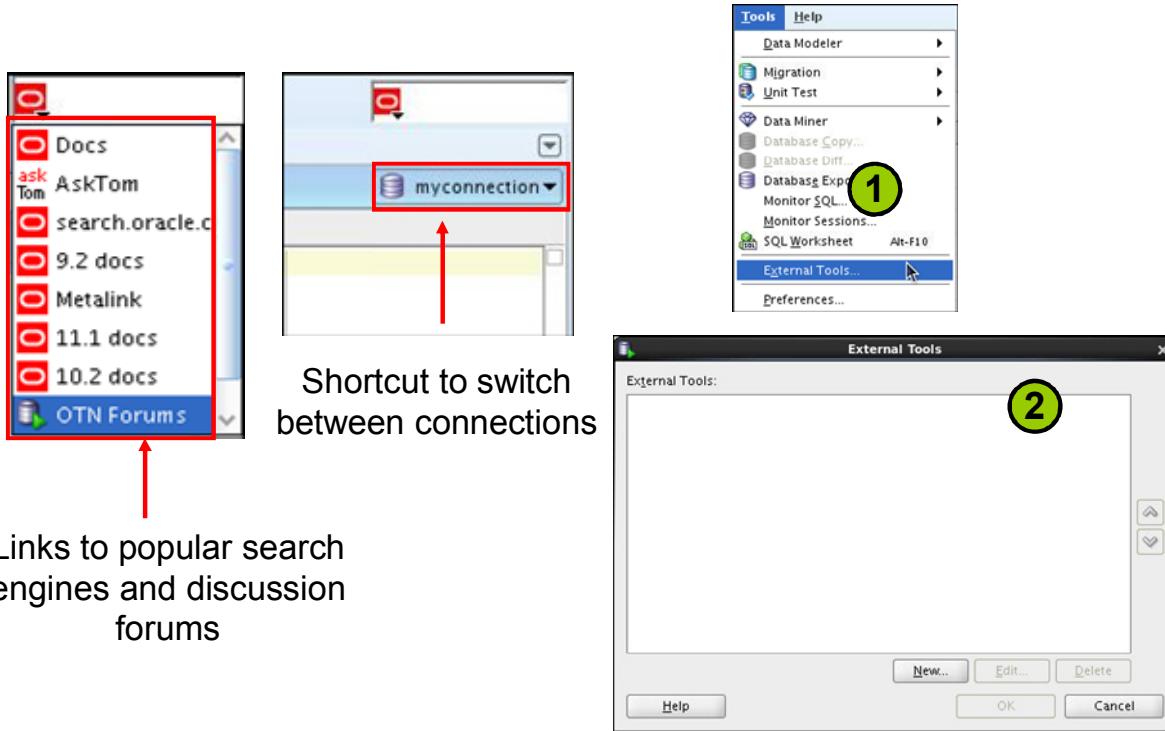
User-defined reports are reports created by SQL Developer users. To create a user-defined report, perform the following steps:

1. Right-click the User Defined Reports node under Reports and select Add Report.
2. In the Create Report dialog box, specify the report name and the SQL query to retrieve information for the report. Then click Apply.

In the example in the slide, the report name is specified as `emp_sal`. An optional description is provided indicating that the report contains details of employees with `salary >= 10000`. The complete SQL statement for retrieving the information to be displayed in the user-defined report is specified in the SQL box. You can also include an optional tool tip to be displayed when the cursor stays briefly over the report name in the Reports navigator display.

You can organize user-defined reports in folders and you can create a hierarchy of folders and subfolders. To create a folder for user-defined reports, right-click the User Defined Reports node or any folder name under that node and select Add Folder. Information about user-defined reports, including any folders for these reports, is stored in a file named `UserReports.xml` in the directory for user-specific information.

Search Engines and External Tools



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

ORACLE

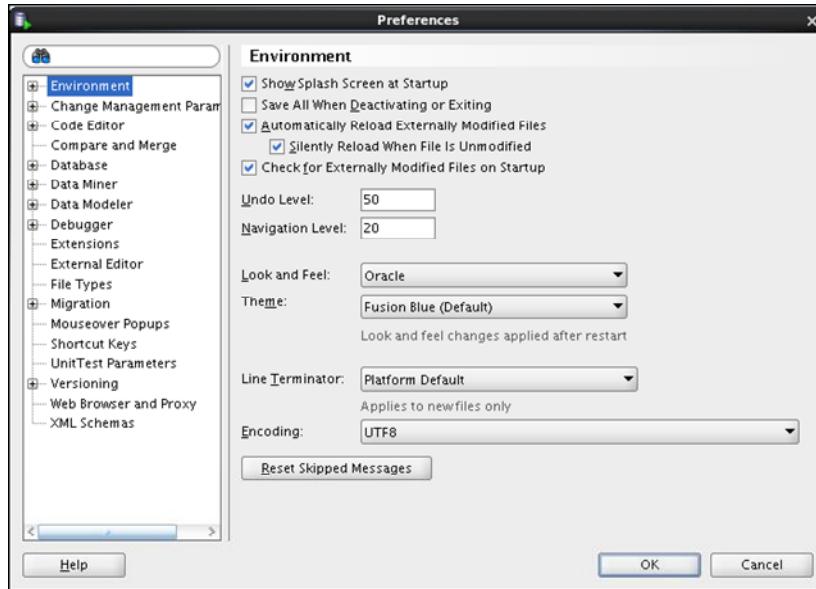
To enhance the productivity of developers, SQL Developer has added quick links to popular search engines and discussion forums such as AskTom, Google, and so on. Also, you have shortcut icons to some of the frequently used tools such as Notepad, Microsoft Word, and Dreamweaver, available to you.

You can add external tools to the existing list or even delete shortcuts to the tools that you do not use frequently. To do so, perform the following steps:

1. From the Tools menu, select External Tools.
2. In the External Tools dialog box, select New to add new tools. Select Delete to remove any tool from the list.

Setting Preferences

- Customize the SQL Developer interface and environment.
- In the Tools menu, select Preferences.



ORACLE

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

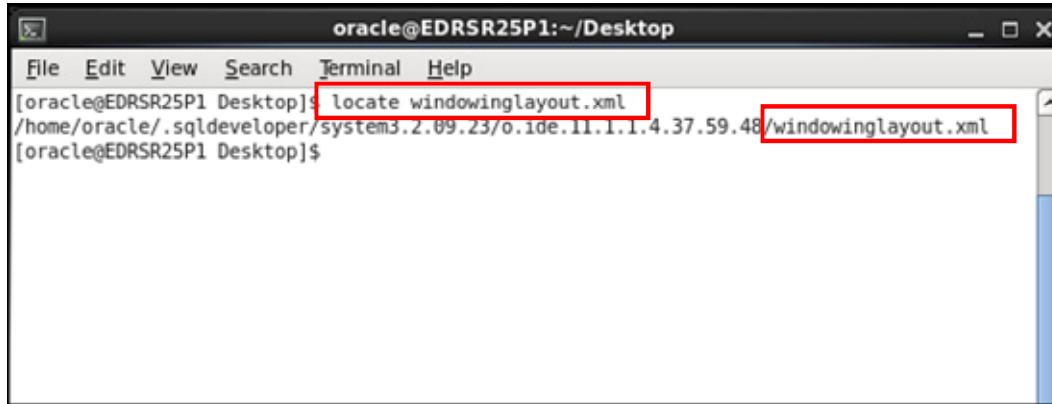
You can customize many aspects of the SQL Developer interface and environment by modifying SQL Developer preferences according to your needs. To modify SQL Developer preferences, select Tools, and then Preferences.

The preferences are grouped into the following categories:

- Environment
- Change Management parameter
- Code Editors
- Compare and Merge
- Database
- Data Miner
- Data Modeler
- Debugger
- Extensions
- External Editor
- File Types
- Migration

- Mouseover Popups
- Shortcut Keys
- Unit Test Parameters
- Versioning
- Web Browser and Proxy
- XML Schemas

Resetting the SQL Developer Layout



A screenshot of a terminal window titled "oracle@EDRSR25P1:~/Desktop". The window shows the following command and its output:

```
[oracle@EDRSR25P1 Desktop]$ locate windowinglayout.xml
/home/oracle/.sqldeveloper/system3.2.09.23/o.ide.11.1.1.4.37.59.48/windowinglayout.xml
[oracle@EDRSR25P1 Desktop]$
```

The command "locate windowinglayout.xml" is highlighted with a red box. The resulting file path "/home/oracle/.sqldeveloper/system3.2.09.23/o.ide.11.1.1.4.37.59.48/windowinglayout.xml" is also highlighted with a red box.

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red horizontal bar.

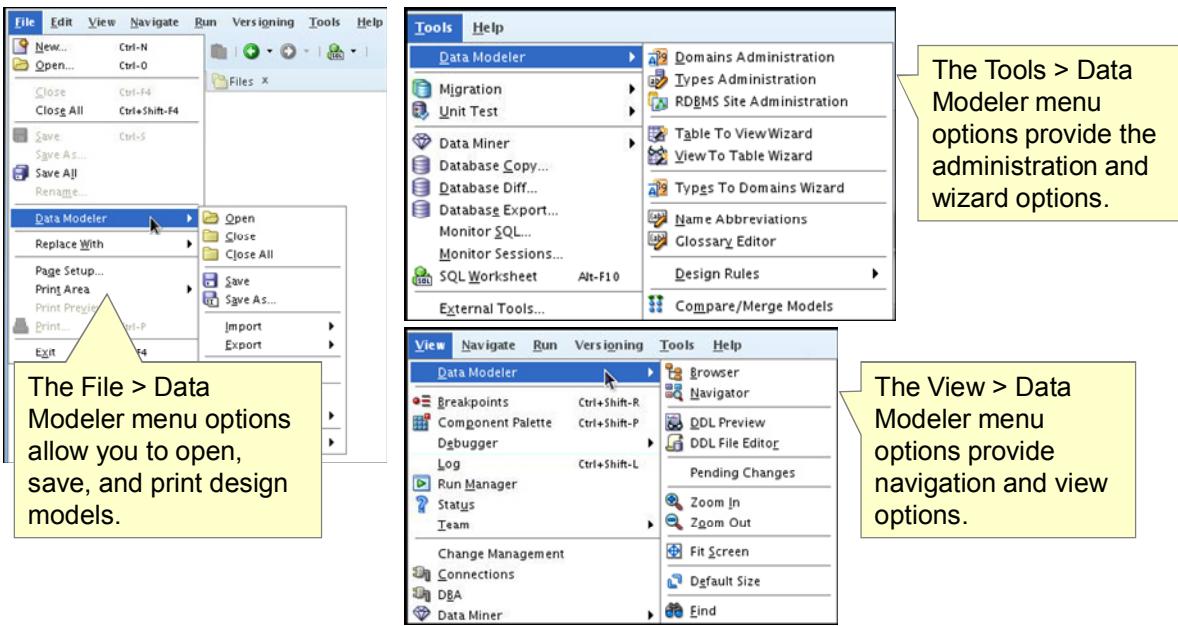
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

While working with SQL Developer, if the Connections Navigator disappears or if you cannot dock the Log window in its original place, perform the following steps to fix the problem:

1. Exit SQL Developer.
2. Open a terminal window and use the locate command to find the location of `windowinglayout.xml`.
3. Go to the directory that has `windowinglayout.xml` and delete it.
4. Restart SQL Developer.

Data Modeler in SQL Developer

SQL Developer includes an integrated version of SQL Developer Data Modeler.



ORACLE

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

Using the integrated version of the SQL Developer Data Modeler, you can:

- Create, open, import, and save a database design
- Create, modify, and delete Data Modeler objects

To display Data Modeler in a pane, click Tools, and then Data Modeler. The Data Modeler menu under Tools includes additional commands, for example, that enable you to specify design rules and preferences.

Summary

In this appendix, you should have learned how to use SQL Developer to do:

- Browse, create, and edit database objects
- Execute SQL statements and scripts in SQL Worksheet
- Create and save custom reports
- Browse the Data Modeling options in SQL Developer



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

SQL Developer is a free graphical tool to simplify database development tasks. Using SQL Developer, you can browse, create, and edit database objects. You can use SQL Worksheet to run SQL statements and scripts. SQL Developer enables you to create and save your own special set of reports for repeated use.

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 JDeveloper

ORACLE

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

Objectives

After completing this appendix, you should be able to:

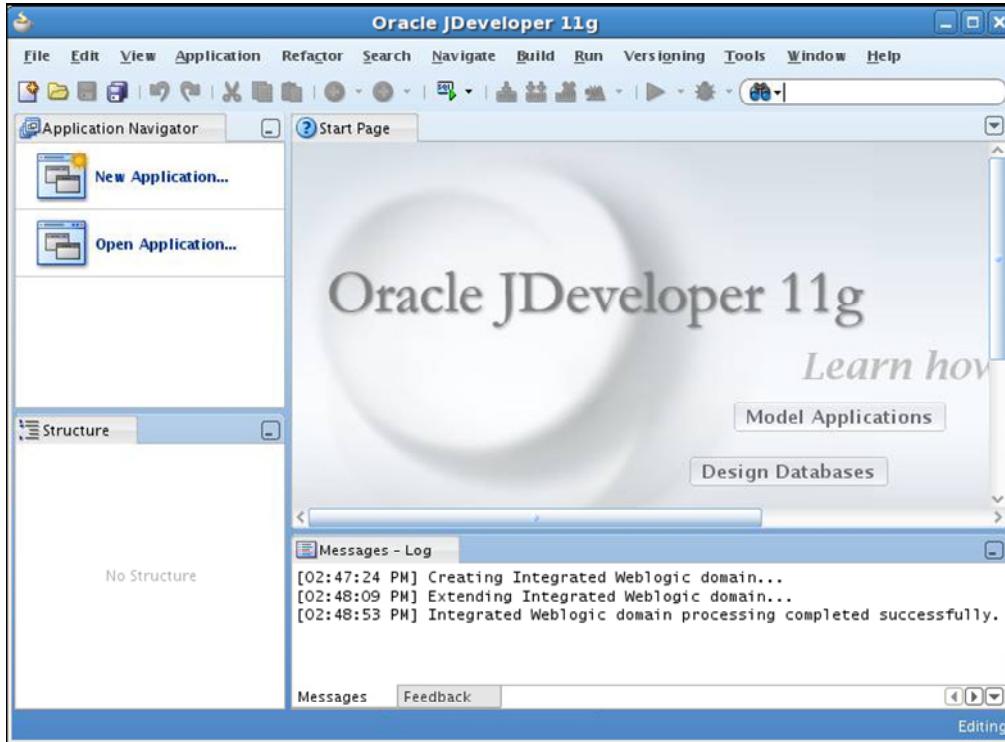
- List the key features of Oracle JDeveloper
- Create a database connection in JDeveloper
- Manage database objects in JDeveloper
- Use JDeveloper to execute SQL commands
- Create and run PL/SQL program units
- Add external tool commands in JDeveloper
- Use, edit, and delete external tool commands in JDeveloper



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

In this appendix, you are introduced to the JDeveloper tool. You learn how to use JDeveloper for your database development tasks.

Oracle JDeveloper

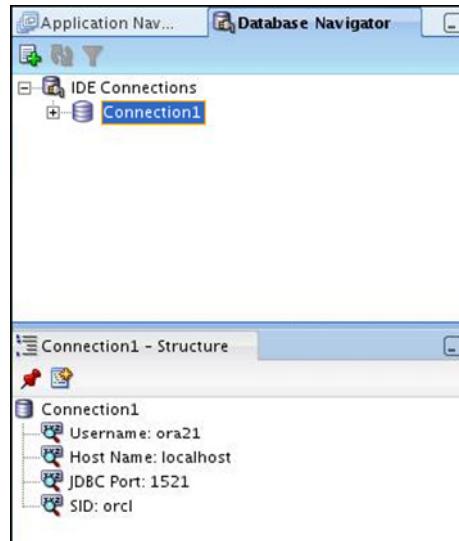


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle JDeveloper is an integrated development environment (IDE) for developing and deploying Java applications and web services. It supports every stage of the software development life cycle (SDLC) from modeling through deploying. It has the features to use the latest industry standards for Java, XML, and SQL while developing an application.

Oracle JDeveloper 11g initiates a new approach to J2EE development with features that enable visual and declarative development. This innovative approach makes J2EE development simple and efficient.

Database Navigator

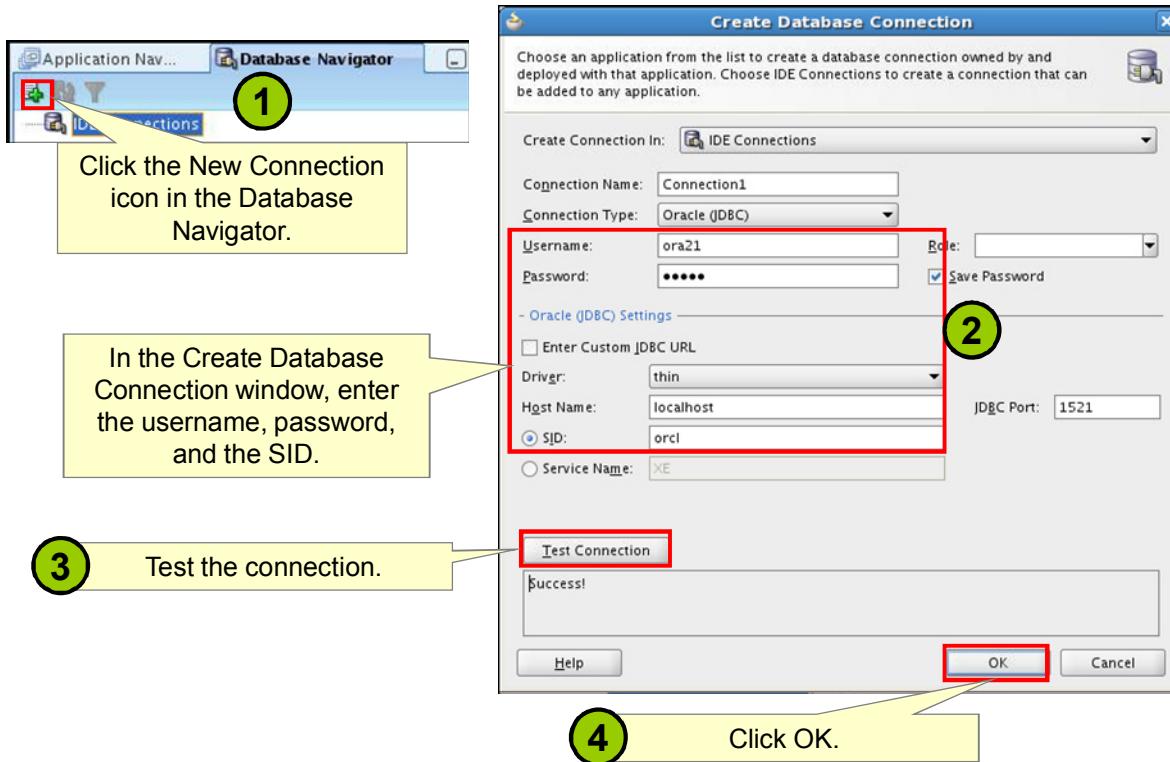


ORACLE

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

Using Oracle JDeveloper, you can store the information that is necessary to connect to a database in an object called “connection.” A connection is stored as part of the IDE settings, and can be exported and imported for easy sharing among groups of users. A connection serves several purposes, from browsing the database and building applications, all the way through to deployment.

Creating a Database Connection



ORACLE

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

A connection is an object that specifies the necessary information for connecting to a specific database as a specific user of that database. You can create and test connections for multiple databases and for multiple schemas.

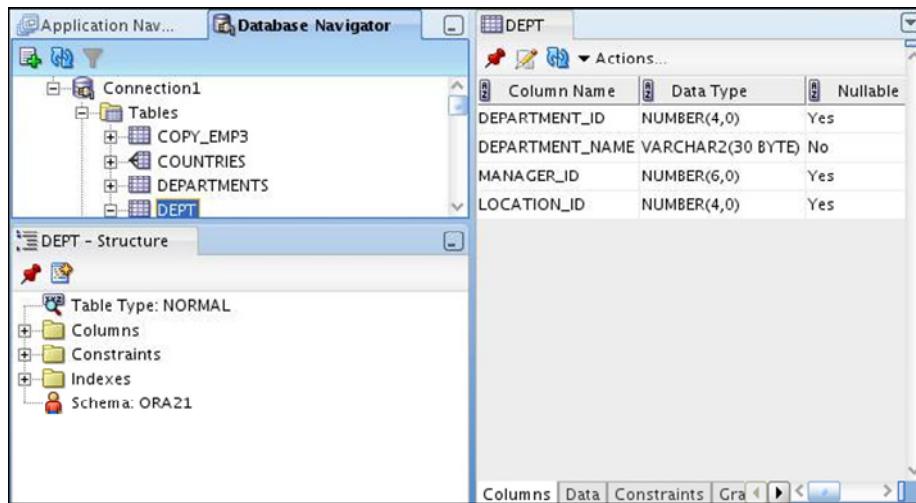
To create a database connection, perform the following steps:

1. Click the New Connection icon in the Database Navigator.
2. In the Create Database Connection window, enter the connection name. Enter the username and password of the schema that you want to connect to. Enter the SID of the database that you want to connect to.
3. Click Test to ensure that the connection has been set correctly.
4. Click OK.

Browsing Database Objects

Use the Database Navigator to:

- Browse through many objects in a database schema
- Review the definitions of objects at a glance



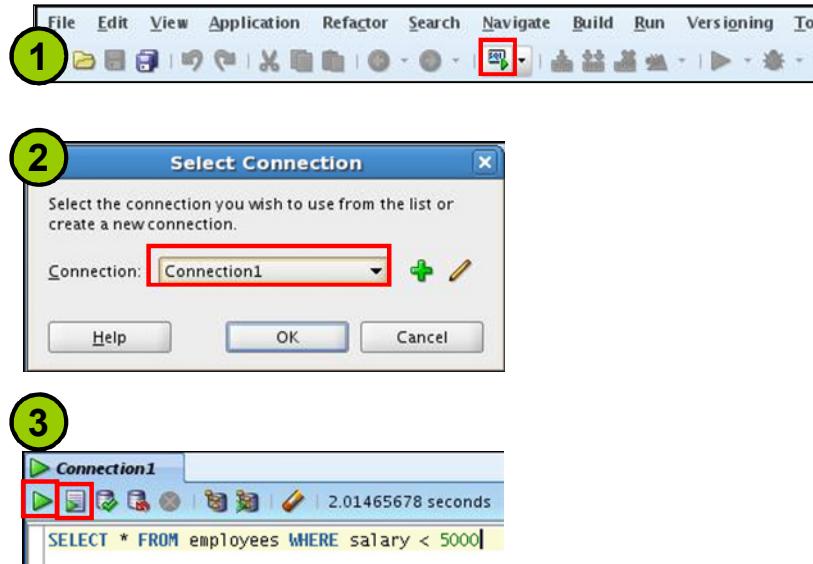
ORACLE

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

After you create a database connection, you can use the Database Navigator to browse through many objects in a database schema, including tables, views, indexes, packages, procedures, triggers, and types.

You can view object definitions, which are broken into tabs of information that is pulled out of the data dictionary. For example, if you select a table in the Navigator, the details about columns, constraints, grants, statistics, triggers, and so on are displayed on an easy-to-read tabbed page.

Executing SQL Statements



ORACLE

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

To execute a SQL statement, perform the following steps:

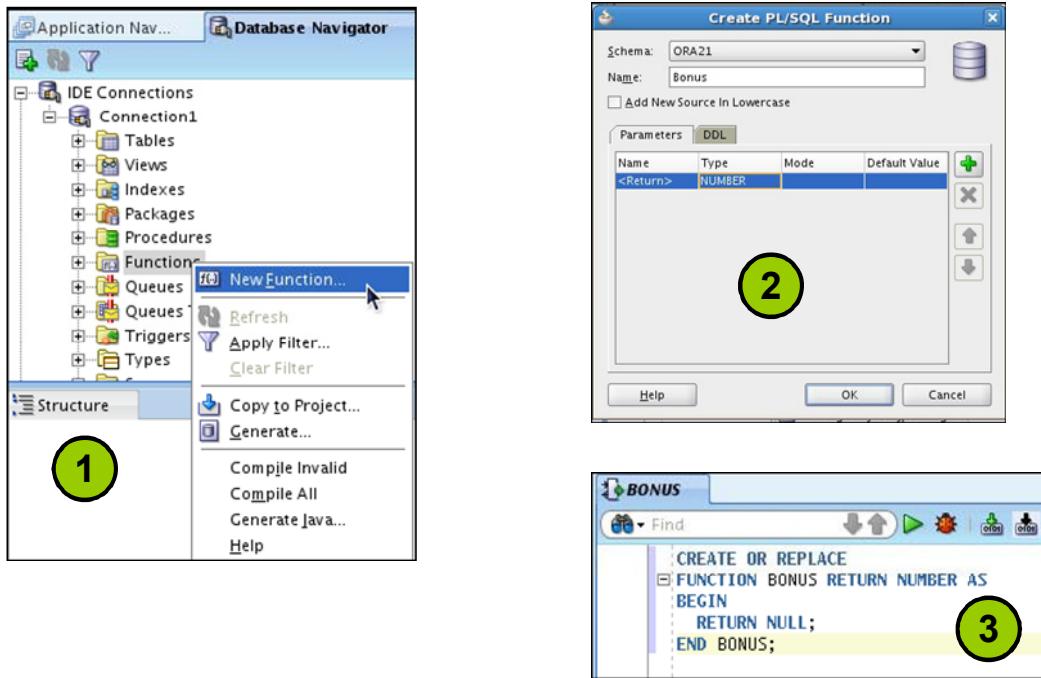
1. Click the Open SQL Worksheet icon.
2. Select the connection.
3. Execute the SQL command by:
 - Clicking the **Execute statement** icon or pressing F9. The output is as follows:

Results			
	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
1	100	Steven	King
2	101	Neena	Kochhar

- Clicking the **Run Script** icon or pressing F5. The output is as follows:

Script Output		
EMPLOYEE_ID	FIRST_NAME	LAST_NAME
100	Steven	King

Creating Program Units



Skeleton of the function

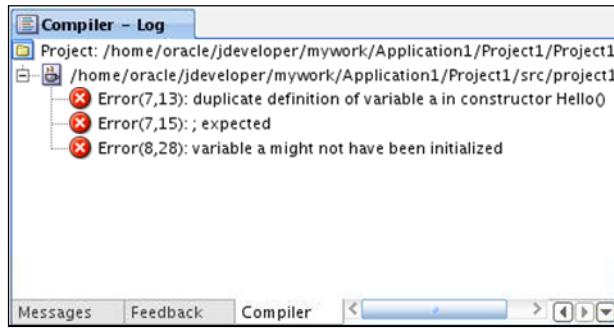
ORACLE

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

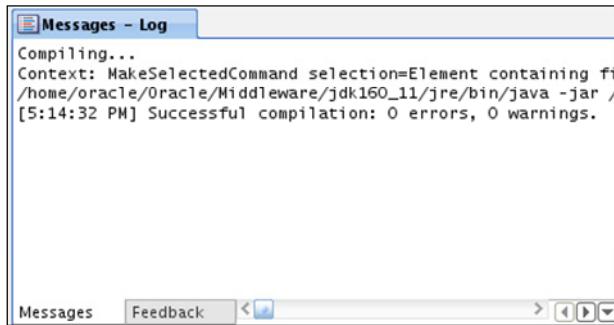
To create a PL/SQL program unit, perform the following steps:

1. Select View > Database Navigator. Select and expand a database connection. Right-click a folder corresponding to the object type (Procedures, Packages, or Functions). Select "New (Procedures, Packages, or Functions)".
2. Enter a valid name for the function, package, or procedure, and click OK.
3. A skeleton definition is created and opened in the Code Editor. You can then edit the subprogram to suit your need.

Compiling



Compilation with errors



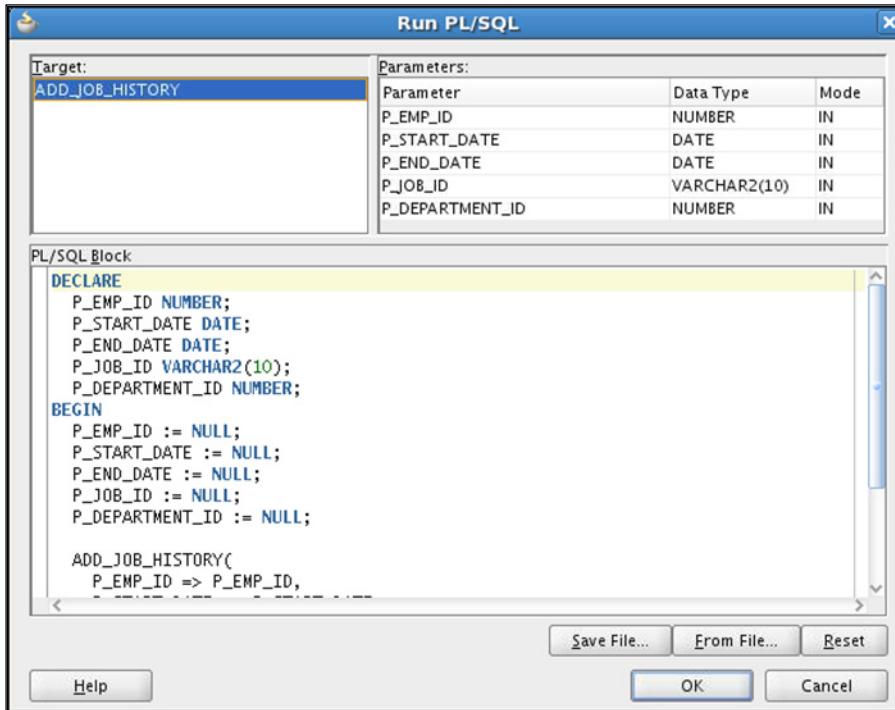
Compilation without errors

ORACLE

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

After editing the skeleton definition, you need to compile the program unit. Right-click the PL/SQL object that you need to compile in the Connection Navigator, and then select Compile. Alternatively, you can press Ctrl + Shift + F9 to compile.

Running a Program Unit

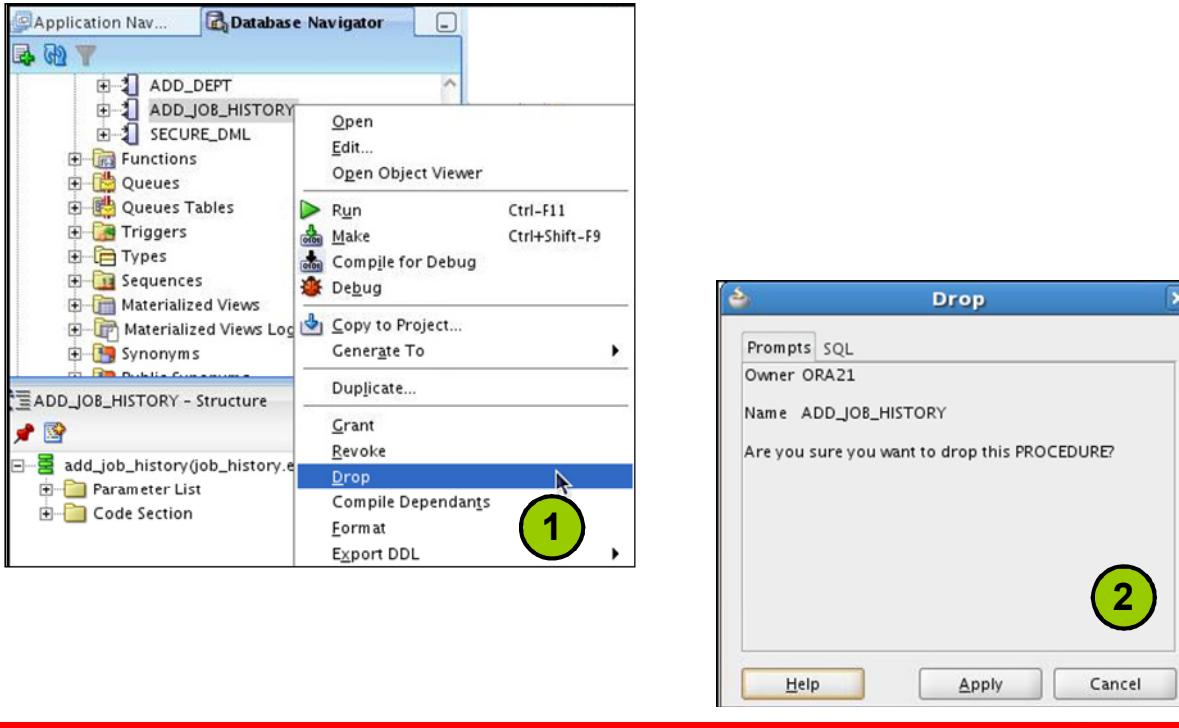


ORACLE

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

To execute the program unit, right-click the object and select Run. The Run PL/SQL dialog box appears. You may need to change the NULL values with reasonable values that are passed into the program unit. After you change the values, click OK. The output is displayed in the Message-Log window.

Dropping a Program Unit



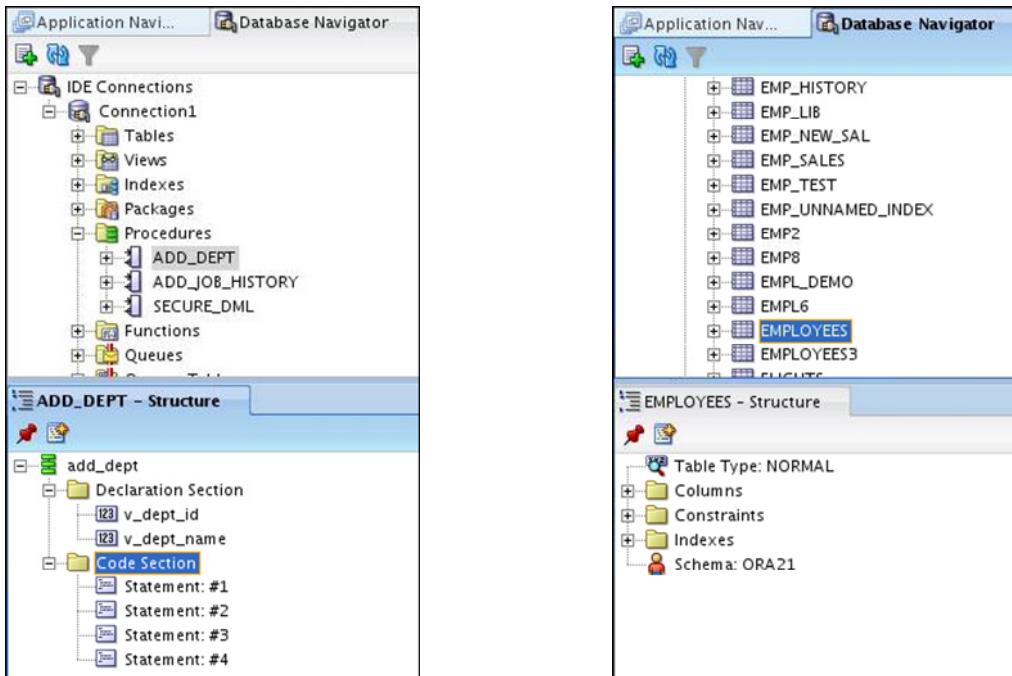
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

To drop a program unit, perform the following steps:

1. Right-click the object and select Drop.
The Drop Confirmation dialog box appears.
2. Click Apply.
The object is dropped from the database.

Structure Window



ORACLE

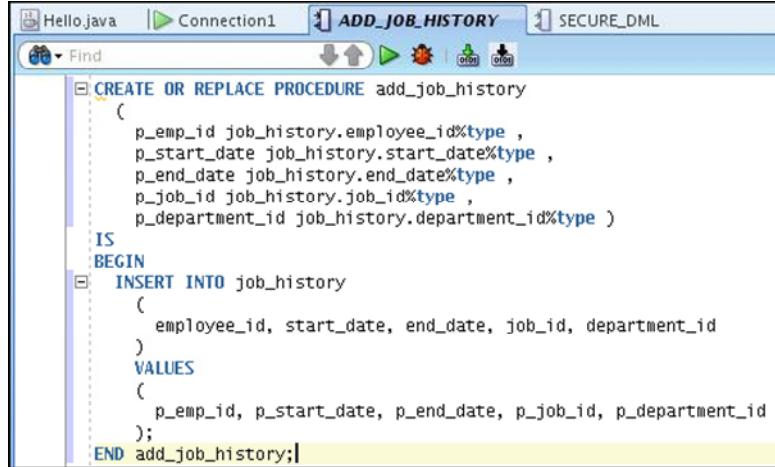
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The Structure window offers a structural view of the data in the document that is currently selected in the active window of those windows that participate in providing structure: the navigators, the editors and viewers, and the Property Inspector.

In the Structure window, you can view the document data in a variety of ways. The structures that are available for display are based on document type. For a Java file, you can view code structure, UI structure, or UI model data. For an XML file, you can view XML structure, design structure, or UI model data.

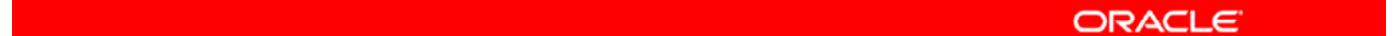
The Structure window is dynamic, tracking always the current selection of the active window (unless you freeze the window's contents on a particular view), as is pertinent to the currently active editor. When the current selection is a node in the navigator, the default editor is assumed. To change the view on the structure for the current selection, click a different structure tab.

Editor Window



The screenshot shows the Oracle SQL Developer interface with an open editor window. The tab bar at the top has tabs for "Hello.java", "Connection1", "ADD_JOB_HISTORY" (which is the active tab), and "SECURE_DML". Below the tabs is a toolbar with icons for Find, Save, Run, and others. The main area contains PL/SQL code:

```
CREATE OR REPLACE PROCEDURE add_job_history
(
    p_emp_id job_history.employee_id%type ,
    p_start_date job_history.start_date%type ,
    p_end_date job_history.end_date%type ,
    p_job_id job_history.job_id%type ,
    p_department_id job_history.department_id%type )
IS
BEGIN
    INSERT INTO job_history
    (
        employee_id, start_date, end_date, job_id, department_id
    )
    VALUES
    (
        p_emp_id, p_start_date, p_end_date, p_job_id, p_department_id
    );
END add_job_history;
```

The Oracle logo is displayed on a red horizontal banner.

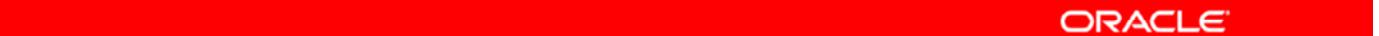
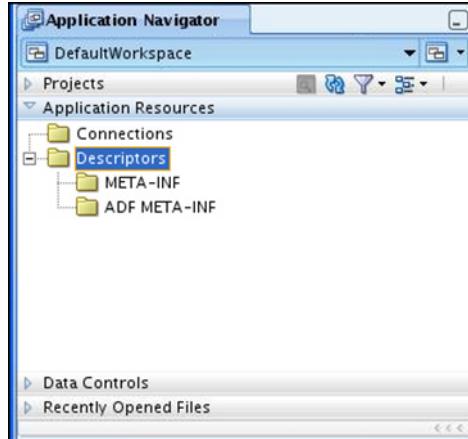
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can view all your project files in one single editor window, open multiple views of the same file, or open multiple views of different files.

The tabs at the top of the editor window are the document tabs. Clicking a document tab gives that file focus, bringing it to the foreground of the window in the current editor.

The tabs at the bottom of the editor window for a given file are the editor tabs. Clicking an editor tab opens the file in that editor.

Application Navigator

ORACLE

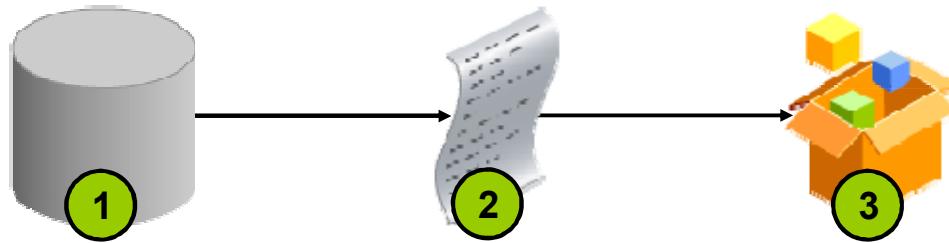
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The Application Navigator gives you a logical view of your application and the data that it contains. The Application Navigator provides an infrastructure that the different extensions can plug in to and use to organize their data and menus in a consistent, abstract manner. Even though the Application Navigator can contain individual files (such as Java source files), it is designed to consolidate complex data. Complex data types such as entity objects, Unified Modeling Language (UML) diagrams, Enterprise JavaBeans (EJB), or web services appear in this navigator as single nodes. The raw files that make up these abstract nodes appear in the Structure window.

Deploying Java Stored Procedures

Before deploying Java stored procedures, perform the following steps:

1. Create a database connection.
2. Create a deployment profile.
3. Deploy the objects.



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

Create a deployment profile for the Java stored procedures, and then deploy the classes and, optionally, any public static methods in JDeveloper using the settings in the profile.

Deploying to the database uses the information provided in the Deployment Profile Wizard and two Oracle Database utilities:

- `loadjava` loads the Java class containing the stored procedures to an Oracle database.
- `publish` generates the PL/SQL call-specific wrappers for the loaded public static methods. Publishing enables the Java methods to be called as PL/SQL functions or procedures.

Publishing Java to PL/SQL

The screenshot shows two windows side-by-side. The left window is titled 'TrimLob.java' and contains Java code for a 'TrimLob' class with a main method. The right window is titled 'TRIMLOBPROC' and contains PL/SQL code for creating or replacing a procedure named 'TRIMLOBPROC'. A green circle labeled '1' is over the Java code, and a green circle labeled '2' is over the PL/SQL code.

```
public class TrimLob
{
    public static void main (String args[]) throws SQLException {
        Connection conn=null;
        if (System.getProperty("oracle.jserver.version") != null)
        {
            conn = DriverManager.getConnection("jdbc:default:connection:");
        }
        else
        {
            DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
            conn = DriverManager.getConnection("jdbc:oracle:thin:scott/tiger");
        }
    }
}
```

```
CREATE OR REPLACE PROCEDURE TRIMLOBPROC
as language java
name 'TrimLob.main(java.lang.String[])';
```

ORACLE

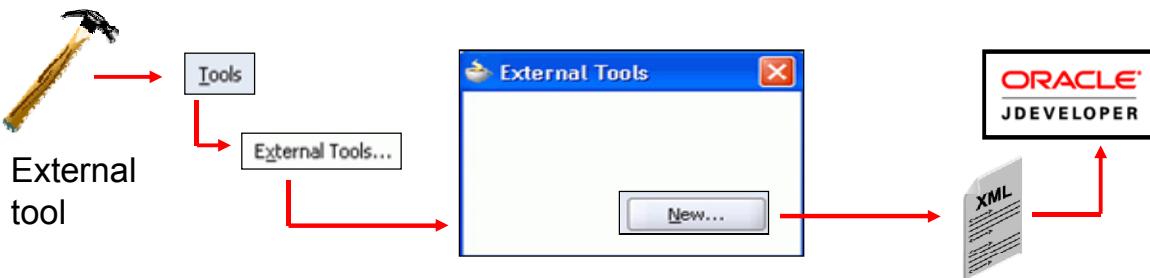
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide shows the Java code and illustrates how to publish the Java code in a PL/SQL procedure.

External Tools with JDeveloper

An external tool:

- Is any script or application that is external to JDeveloper
 - Command-line or graphical applications
 - Operating system commands or scripts
- Is added to JDeveloper Tools or context menus
- Is configured by using the External Tools dialog box that is opened by selecting Tools > External Tools



The configuration is saved in `product-preferences.xml`.



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

Oracle JDeveloper is an integrated development environment (IDE). Therefore, most of the development tasks that you perform must be possible through the JDeveloper interface.

If JDeveloper does not provide built-in tools or extensions to facilitate a particular development task, you can configure JDeveloper to invoke an appropriate external program or tool to perform the task.

Any application, command-line or graphical in nature, that is external to JDeveloper can be invoked by using the JDeveloper **Tools** menu, context menus, or toolbar.

To add an external tool in JDeveloper, perform the following steps:

1. Select **Tools** > **External Tools**.
2. In the **External Tools** dialog box, click **New**.

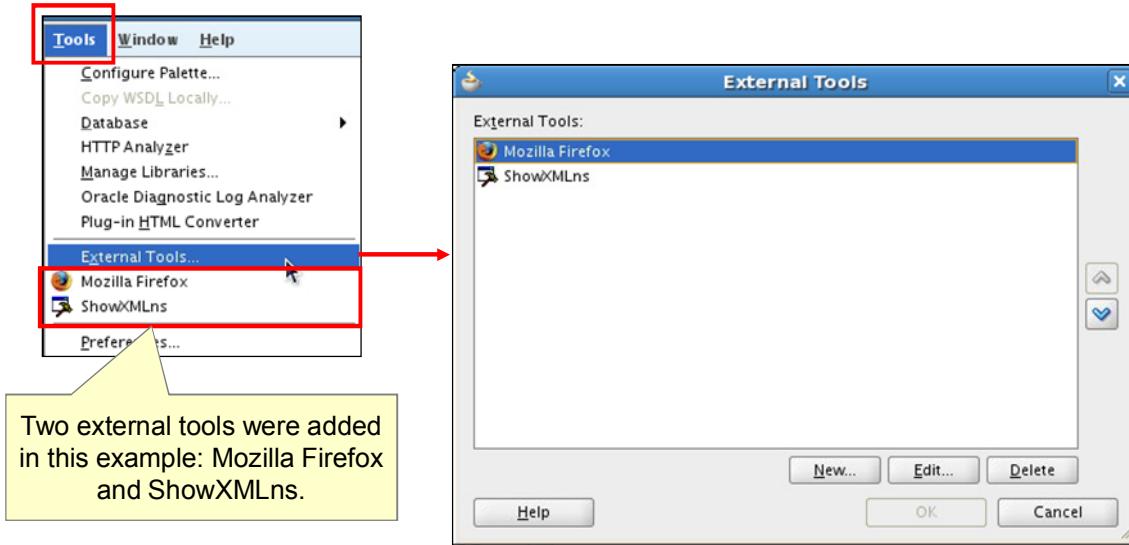
When you click **New**, a wizard guides you through the steps to configure JDeveloper to invoke the external tool. The steps to add an external tool are shown in this appendix.

Note: In Linux, JDeveloper saves the configuration of external tools in a file called `product-preferences.xml`, which is created in the `/home/oracle/.jdeveloper/system11.1.1.2.36.55.36/o.jdeveloper/product-preferences.xml` folder.

Managing External Tools

You can use the External Tools dialog box to:

- Add new tools
- Edit or delete existing tools



ORACLE

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

The **External Tools** dialog box appears when you select **Tools > External Tools**.

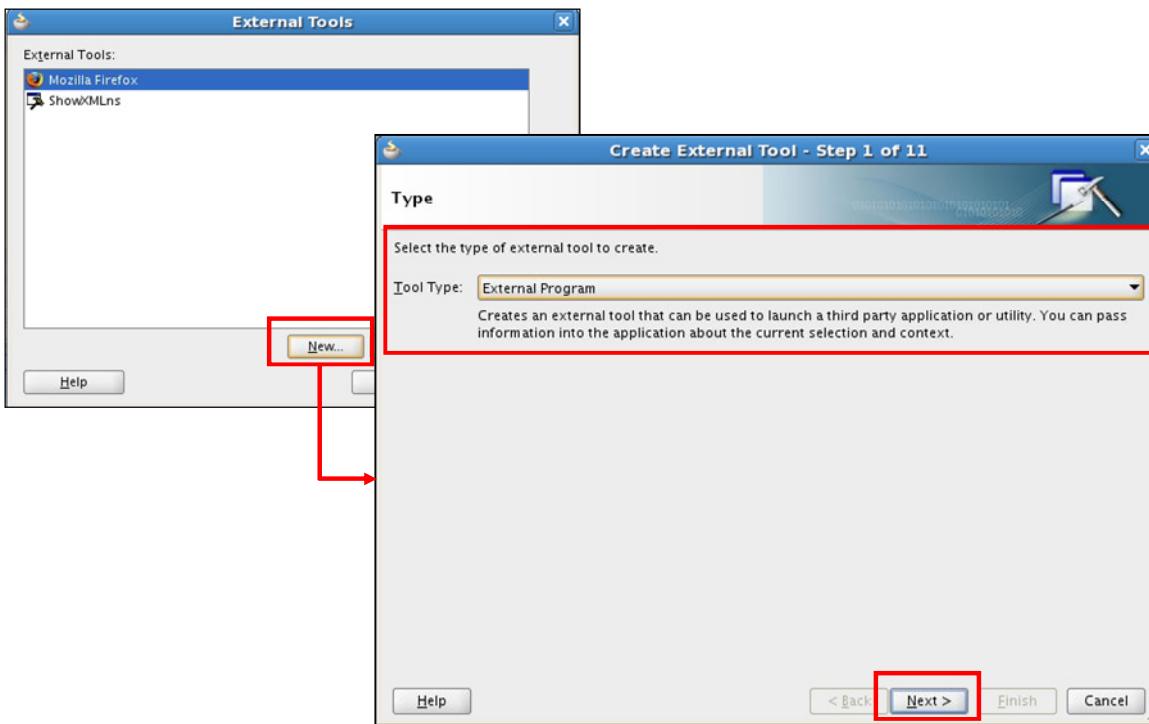
The slide shows the **External Tools** dialog box with two external tools that were added for this course: Mozilla Firefox and ShowXMLNs.

In the **External Tools** dialog box, you can perform the following steps:

1. Click **New** to create a new external tool.
2. Click **Edit** after selecting the tool name to alter the settings for the external tool.
3. Click **Delete** after selecting the tool name to delete the external tool from the JDeveloper environment.

These operations are explained in the next few slides.

Creating a New External Tool: Step 1 of 11 – Type



ORACLE

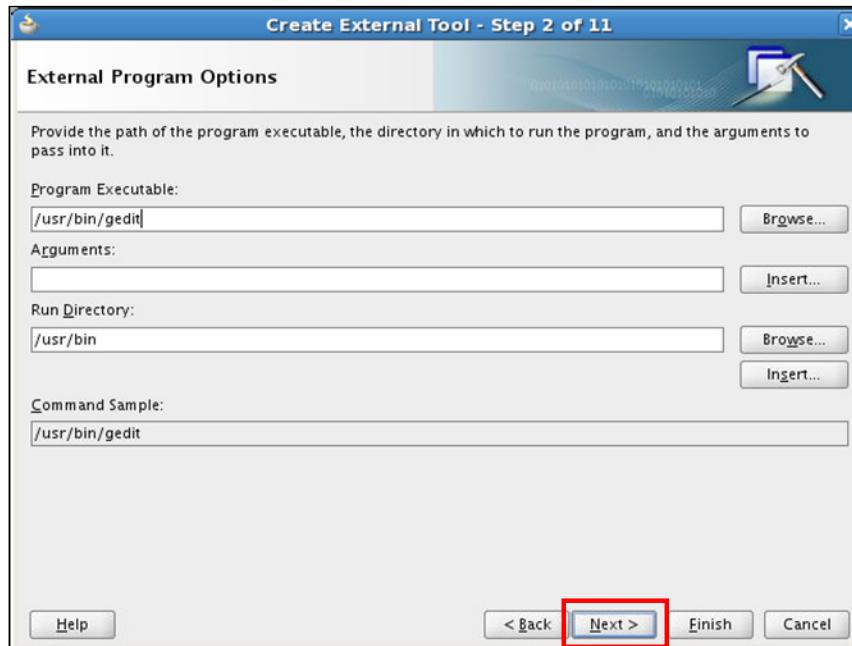
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

After clicking **New** in the **External Tools** dialog box, you are guided through the required steps by the **Create External Tool** wizard.

In **Step 1 of 11: Type**, you select the tool type, and then click **Next**. The “Creating a New External Tool: Step 2 of 11 - External Program Options” window appears.

Note: The number of wizard steps that you will go through depends on the options that you select in the various windows.

Creating a New External Tool: Step 2 of 11 – External Program Options



ORACLE

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

This step enables you to provide the path of the program executable, the directory in which to run the program, and the arguments to pass into it.

Program Executable: Enter the complete path and file name of the external program to be invoked, or click **Browse** to locate it.

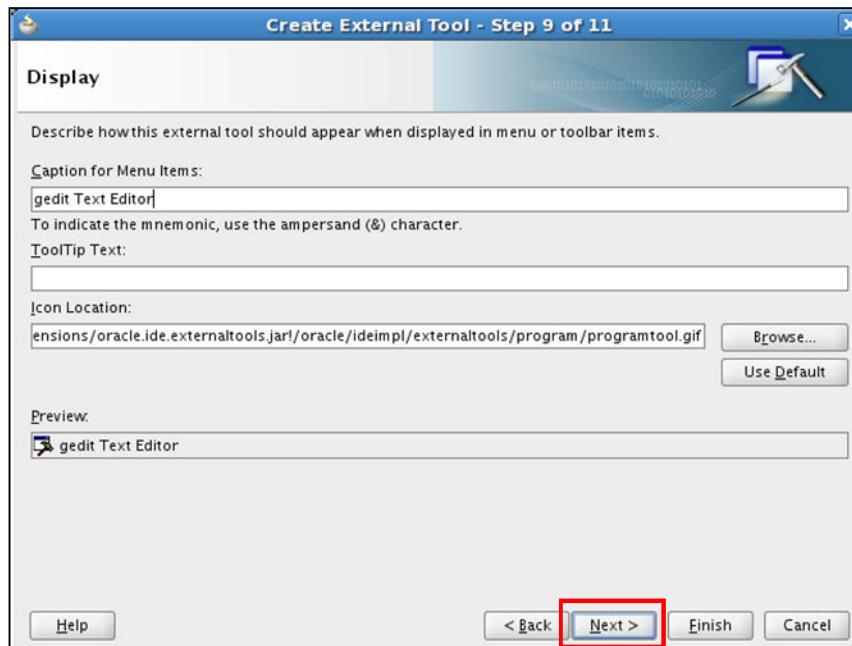
Arguments: Provide arguments that JDeveloper passes to the external program when it is launched. You can add your own arguments, or select one from a list of macros by clicking **Insert**. Make sure that you insert a space between the text you enter and the macros you add.

Run Directory: Enter the complete path name of the directory that the launched external program will use as its current directory, or click **Browse** to locate a directory and select it. You can add your own arguments, or select one from a list of macros by clicking **Insert**. If this field is left blank, the external program will run from the parent directory of the program executable.

Command Sample: After you have supplied values, JDeveloper displays a sample of the command that launches the external program for your inspection.

When all the settings are complete, click **Next**. The "Creating a New External Tool: Step 9 of 11 – Display" window appears.

Creating a New External Tool: Step 9 of 11 – Display



ORACLE

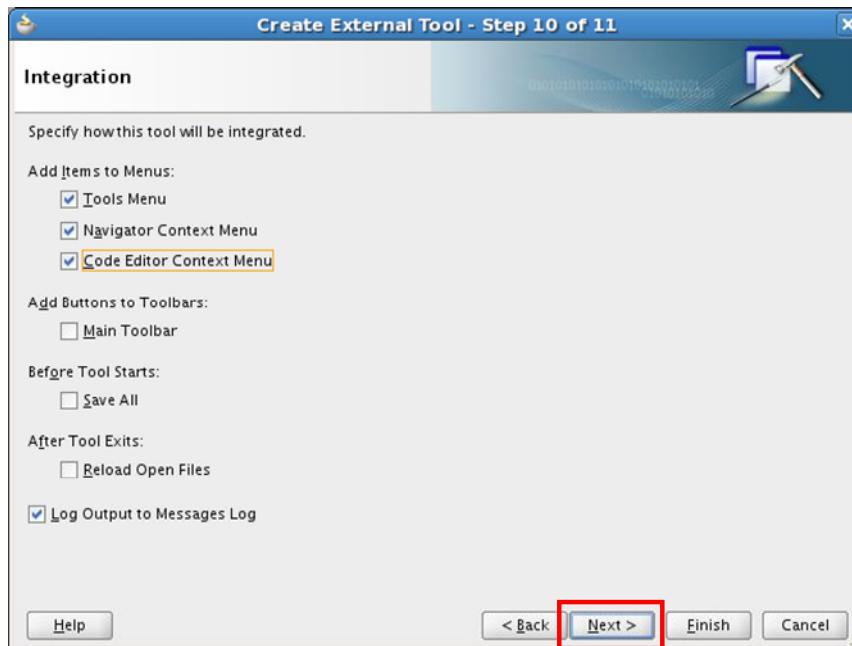
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This step controls the appearance of the menu items and the toolbar buttons that launch the external program from JDeveloper.

- **Caption for Menu Items:** Enter the external program name that will appear in the menus and toolbars.
- **ToolTip Text:** Enter the tool tip label for the external program icon in the JDeveloper IDE toolbar. If you leave this field empty, the tool tip text of the toolbar icon will be the same as "Caption for Menu Items."
- **Icon Location:** If the external program is to be invoked from the toolbar, or if you want the external program menu item to display an icon, enter the path name to an image file or click **Browse** to locate and select it.
- Click **Use Default** to reset the icon location back to the default for the tool. For external program tools, this is the icon of the application being launched on Windows, or a generic external program icon on other platforms.
- **Preview:** This displays the menu entry that you have specified for your review.

When all the settings are complete, click **Next**. The "Creating a New External Tool: Step 10 of 11 – Integration" window appears.

Creating a New External Tool: Step 10 of 11 – Integration



ORACLE

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

This step controls how the external program is integrated into the JDeveloper IDE.

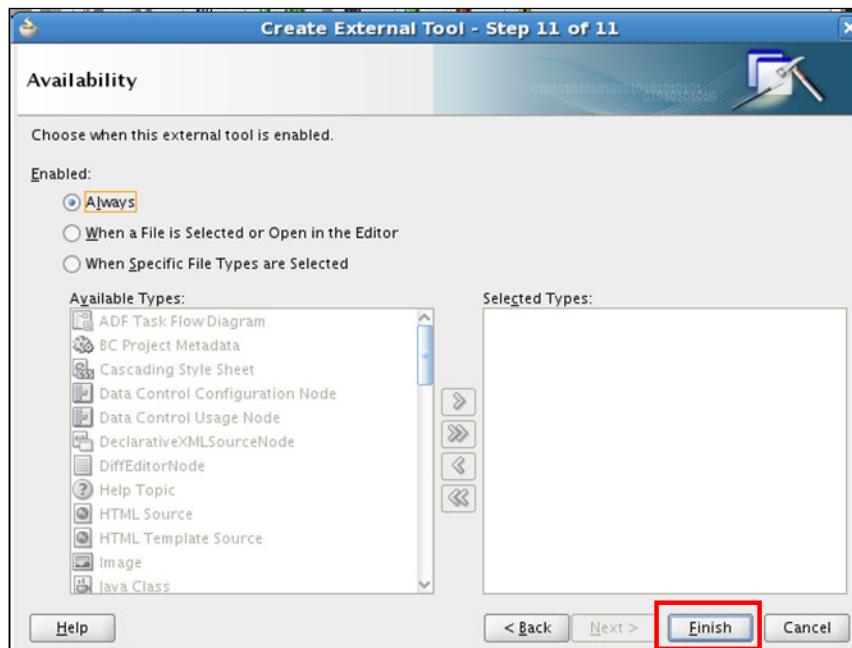
- **Add Items to Menu:** Select options to control which menus display the menu items that launch the external programs.
- **Tools Menu:** Select to place the item in the main Tools menu, under External Tools.
- **Navigator Context Menu:** Select to place the item at the bottom of the context menu (which appears when you right-click) for a document icon in the Navigator.
- **Code Editor Context Menu:** Select to place the item at the bottom of the context menu (which appears when you right-click) for a document window.

Add Buttons to Toolbar: Select options to control where the buttons that launch the external programs will display. Select the **Main Toolbar** check box to place the item at the extreme right of the main window's toolbar.

Before Tool Starts: Select the **Save All** check box to save all open files before starting this tool. **After Tool Exits:** Select the **Reload Open Files** check box to ensure that JDeveloper refreshes any open files after the external program exits.

Log Output to Message Log: Select this check box to send the output of the external tool to the JDeveloper Message Log. When all settings are complete, click **Next**. The "Creating a New External Tool: Step 11 of 11 – Availability" window appears.

Creating a New External Tool: Step 11 of 11 – Availability



ORACLE

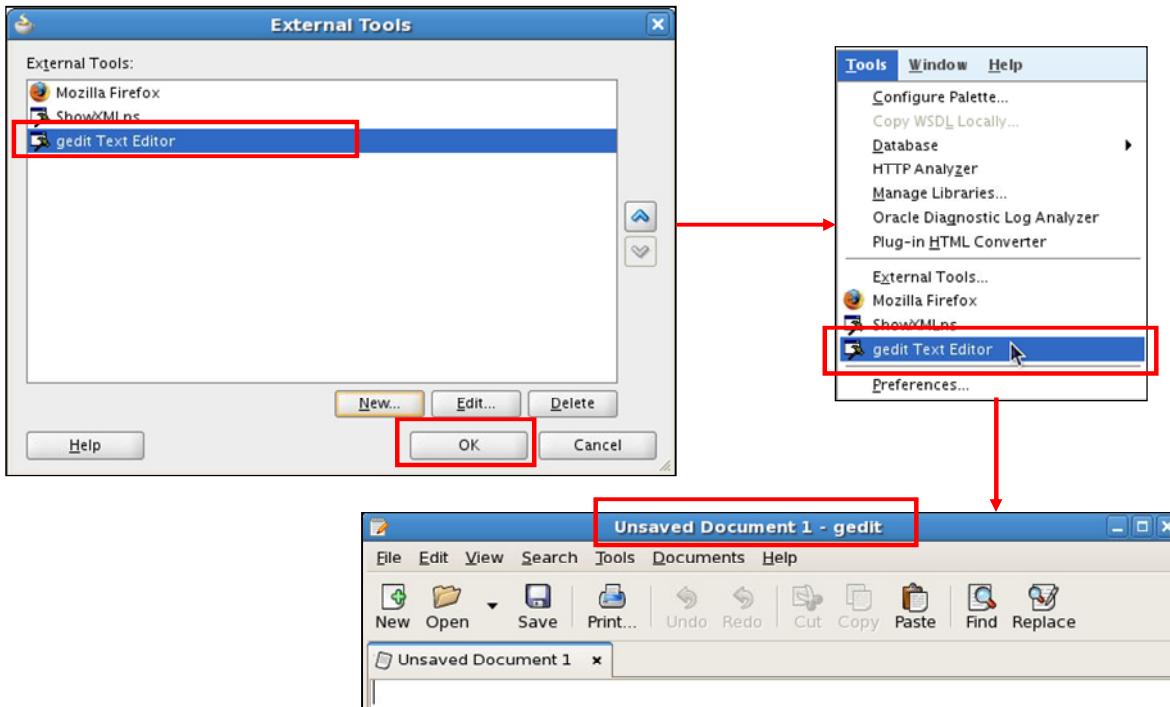
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This step controls when the external program is available from JDeveloper:

- **Always:** Select this option to enable the external program to be launched at all times.
- **When a File is Selected or Open in the Editor:** Select this option to enable the external program to be launched only when a file is selected or open in an editor.
- **When Specific File Types are Selected:** If you select this option, the external program will be available only when the file types that you specified are selected. Select the appropriate file types from Available Types and move them to Selected Types. You can select multiple file types.

When all the settings are complete, click **Finish**. The **External Tools** dialog box appears again.

Using the Newly Added “gedit Text Editor” External Tool



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

ORACLE

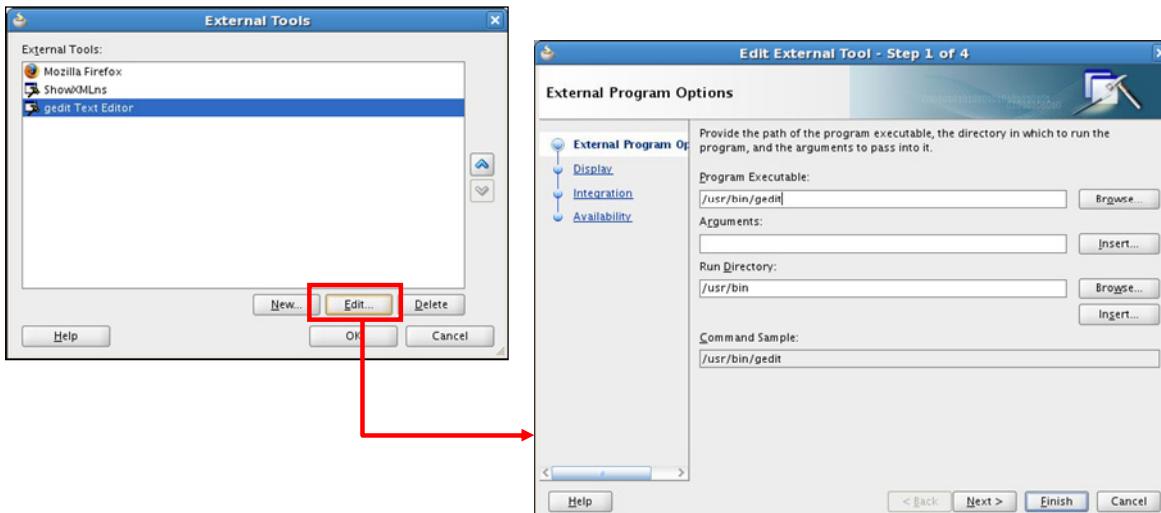
The **External Tools** dialog box now displays the newly added “gedit Text Editor” tool. Click **OK** to exit this dialog box.

Click the **Tools** menu to display the newly added external tool. Select the **gedit Text Editor** menu option to launch the **gedit** text editor as shown in the slide.

Editing an External Tool

In the External Tools dialog box, perform the following steps:

1. Select the tool name.
2. Click Edit.



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

To edit an external tool, in the **External Tools** dialog box, perform the following steps:

1. Select the name of the tool that you want to edit.
2. Click **Edit**.

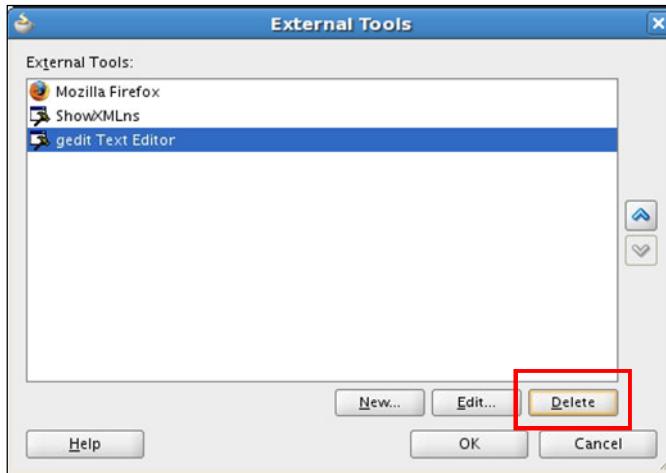
The **Edit External Tool – Step 1 of 4** window appears. Note that the number of wizard steps you go through depends on the external tool that you select.

In the slide example, you want to edit the newly added “gedit Text Editor” external tool. The window displays a graphic representation of the steps of the wizard that you can edit: External Program Options, Display, Integration, and Availability.

Deleting an External Tool

In the External Tools dialog box, perform the following steps:

1. Select the tool name.
2. Click Delete.



ORACLE

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

To remove an external tool, in the **External Tools** dialog box, perform the following steps:

1. Select the name of the tool to be removed.
2. Click **Delete**.

Note: JDeveloper does not prompt for confirmation before the tool entry is deleted. It is recommended that you make a copy of the `product-preferences.xml` file before removing or editing a command.

How Can I Learn More About JDeveloper 11g ?

Topic	Website
Oracle JDeveloper Product Page	http://www.oracle.com/technology/products/jdev/index.html
Oracle JDeveloper 11g Tutorials	http://www.oracle.com/technology/obe/obe11jdev/11/index.html
Oracle JDeveloper 11g Product Documentation	http://www.oracle.com/technology/documentation/jdev.html
Oracle JDeveloper 11g Discussion Forum	http://forums.oracle.com/forums/forum.jspa?forumID=83



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

Summary

In this appendix, you should have learned to:

- List the key features of Oracle JDeveloper
- Create a database connection in JDeveloper
- Manage database objects in JDeveloper
- Use JDeveloper to execute SQL Commands
- Create and run PL/SQL program units
- Add external tool commands in JDeveloper
- Use, edit, and delete external tool commands in JDeveloper



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

PL/SQL API for XMLType

ORACLE

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

Objectives

After completing this lesson, you should be able to:

- Use DBMS_XMLSTORE to store XML data in relational tables
- Describe the Document Object Model (DOM)
- List the W3C recommendations for the DOM
- Use DBMS_XMLDOM to create and modify DOM documents
- Use DBMS_XMLPARSER to access the contents and structure of XML documents
- Use DBMS_XSLPROCESSOR to transform XML documents



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

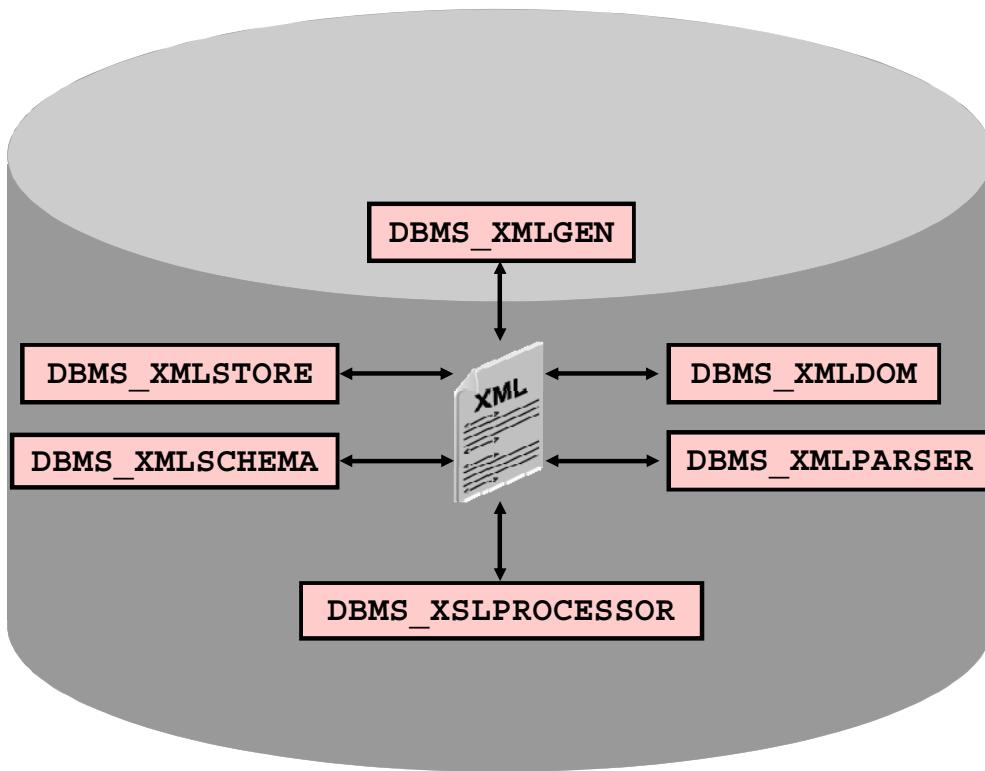
Lesson Agenda

- PL/SQL APIs for XMLType
- Using DBMS_XMLSTORE
- Document Object Model (DOM)
- Using DBMS_XMLDOM to:
 - Create a DOM document
 - Modify a DOM document
 - Add nodes to a DOM document
- Using DBMS_XMLPARSER
- Using DBMS_XSLPROCESSOR



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

PL/SQL API Packages for XMLType



ORACLE

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

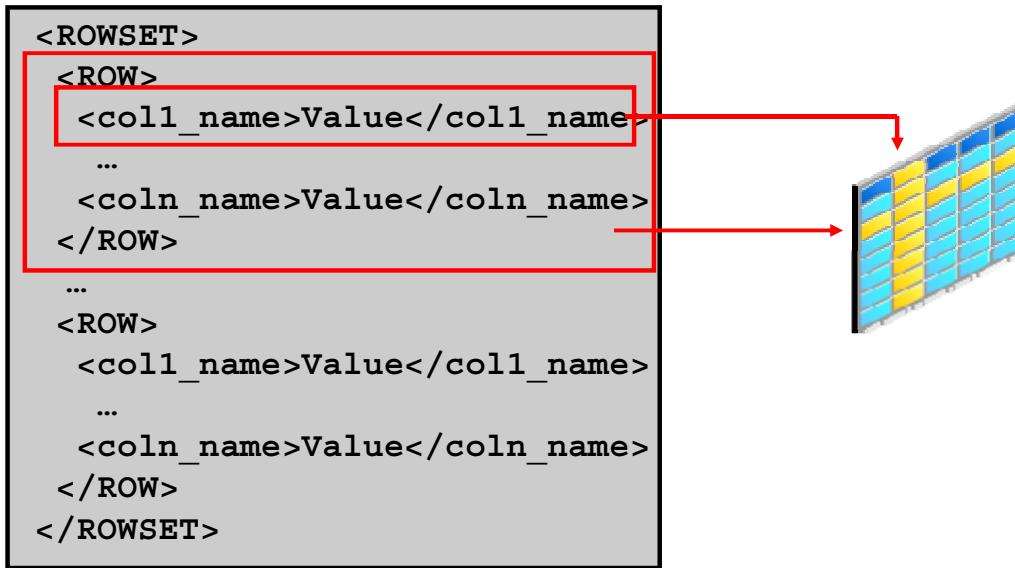
The Oracle XML DB APIs contain various packages that facilitate working with XMLType. You can use the functionality provided by these packages when you create XMLType tables, columns, and views. These packages also provide methods for accessing and manipulating XMLType data.

- DBMS_XMLGEN converts the results of a SQL query to a canonical XML format.
- DBMS_XMLSHEMA provides procedures to register and delete XML schemas.
- DBMS_XMLSTORE enables you to store XML data in relational tables.
- DBMS_XMLDOM controls the access to XMLType objects. This package implements the DOM, an API for HTML and XML documents.
- DBMS_XMLPARSER parses an XMLType object into a DOM tree.
- DBMS_XSLPROCESSOR contains functionality access to the contents and structure of XML documents.

In this lesson, you learn in detail about the use of DBMS_XMLSTORE, DBMS_XMLDOM, DBMS_XMLPARSER, and DBMS_XSLPROCESSOR.

DBMS_XMLSTORE PL/SQL Package

Oracle canonical XML format:



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

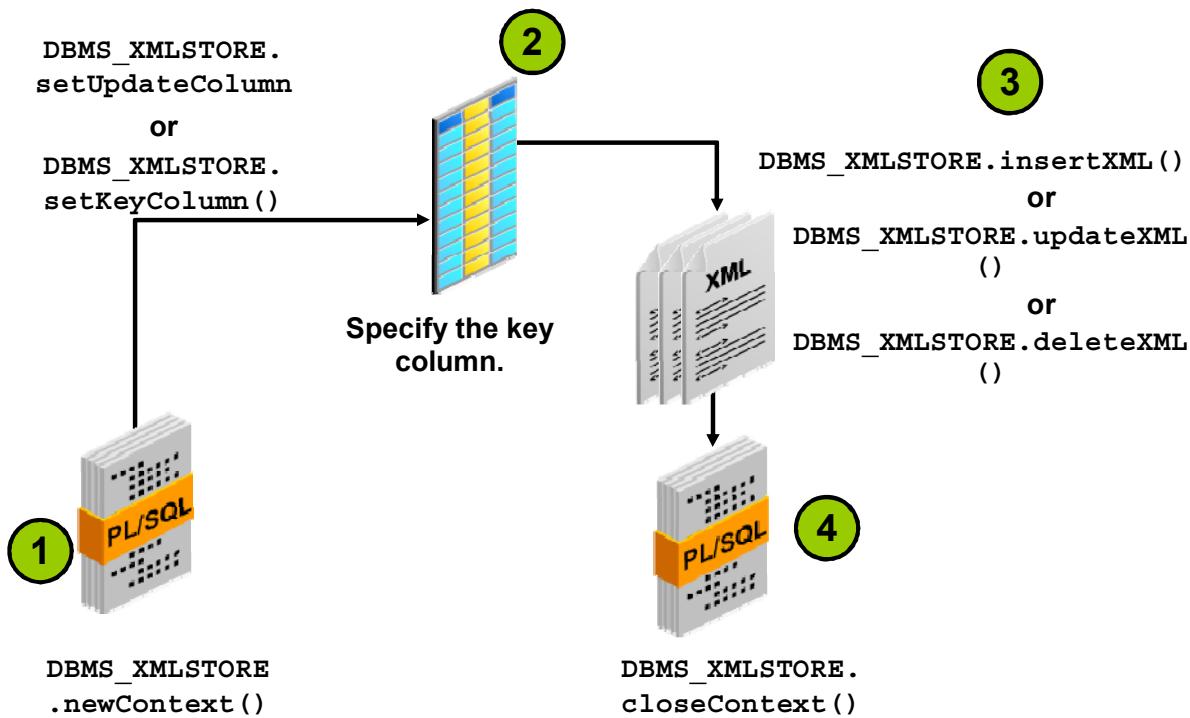
DBMS_XMLSTORE is a PL/SQL package that enables you to perform data manipulation language (DML) operations such as INSERT, UPDATE, and DELETE, on relational tables by using XML. This PL/SQL package allows input of XMLType, CLOB, and VARCHAR data.

To use the DBMS_XMLSTORE PL/SQL package, you must format your XML documents in the Oracle canonical XML format as shown in the slide.

In this format, each child element of a `ROW` element maps to a column in the table to be updated and the element name is the column name. The `ROW` element in the XML document maps to a row in the table. All the `ROW` elements are placed under the `ROWSET` element. The DBMS_XMLSTORE PL/SQL package takes this canonical XML mapping, converts it into object-relational constructs, and inserts, updates, or deletes the value in the relational table.

Note: The output shown in this slide is produced by executing the queries in SQL*Plus.

Using the DBMS_XMLSTORE PL/SQL Package: Steps



ORACLE

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

To use the DBMS_XMLSTORE PL/SQL package, perform the following steps:

1. Create a context handle by calling the `DBMS_XMLSTORE.newContext` PL/SQL function and supplying it with the table name to use for the DML operations. You can ensure case-sensitivity by using double quotation marks around the string that is passed to the PL/SQL function. The default behavior is that XML documents identify rows with the `<ROW>` tag.
You can override this default behavior by calling the `setRowTag` PL/SQL function.
2. Specify the key columns for the DML operations.
 - a. **For inserts:** The default behavior is to insert values for all columns whose corresponding elements are present in the XML document. Alternatively, for performance benefits, you can set the list of columns to insert by calling the `DBMS_XMLSTORE.setUpdateColumn` PL/SQL function for each column.
 - b. **For updates:** You must specify one or more (pseudo) key columns by using the `DBMS_XMLSTORE.setKeyColumn` PL/SQL function. These key columns are used to specify the rows to be updated. The key columns need not be actual keys of the table. As long as they uniquely specify a row, they can be used with `setKeyColumn`.

The default behavior is to update all the columns in the rows identified by `setKeyColumn`, whose corresponding elements are present in the XML document.

Alternatively, for performance benefits, you can specify the list of columns to be updated by using the `DBMS_XMLSTORE.setUpdateColumn` PL/SQL function.

- c. **For deletions:** You must specify (pseudo) key columns to identify the rows to delete by using the `DBMS_XMLSTORE.setKeyColumn` PL/SQL function.
- 3. Provide a document to `insertXML`, `updateXML`, or `deleteXML`. You can repeat this step multiple times with several XML documents.
- 4. Close the context with the `DBMS_XMLSTORE.closeContext` PL/SQL function.

Inserting with DBMS_XMLSTORE

```

BEGIN
    insCtx := DBMS_XMLSTORE.newContext('HR.NEW_EMPLOYEES');
    1
    DBMS_XMLSTORE.clearUpdateColumnList(insCtx);
    2
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'EMPLOYEE_ID');
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'SALARY');
    3
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'HIRE_DATE');
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'DEPARTMENT_ID');
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'JOB_ID');
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'EMAIL');
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'LAST_NAME');
    rows := DBMS_XMLSTORE.insertXML(insCtx, xmlDoc);
    4
    DBMS_OUTPUT.put_line(rows || ' rows inserted.');
    DBMS_XMLSTORE.closeContext(insCtx);
    5
END;
/

```

ORACLE

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

The slide example shows how to use DBMS_XMLSTORE to insert information for two new employees into the NEW_EMPLOYEES table. The information is provided in the form of XML data. The NEW_EMPLOYEES table is created for illustration purpose as follows:

```
CREATE TABLE NEW_EMPLOYEES AS SELECT * FROM EMPLOYEES;
```

To insert an XML document into a table or view, you must supply the table or view name and the document. DBMS_XMLSTORE parses the document, and then creates an INSERT statement into which it binds all the values. In the example in the slide, you can see calls to the DBMS_XMLSTORE PL/SQL package to accomplish the insert.

1. newContext creates a saved context by using the NEW_EMPLOYEES table.
2. clearUpdateColumnList clears the update settings.
3. setUpdateColumn sets the columns to be updated as a list of values.
4. insertXML inserts the XML document into the NEW_EMPLOYEES table and returns the number of rows inserted.
5. closeContext closes the context.

Before the insertion, you can query the NEW_EMPLOYEES table as follows:

```
SELECT employee_id, salary, hire_date, job_id, email, last_name  
FROM new_employees  
WHERE department_id = 50;
```

EMPLOYEE_ID	SALARY	HIRE_DATE	JOB_ID	EMAIL	LAST_NAME
198	2600	21-JUN-99	SH_CLERK	DOCONNEL	OConnell
199	2600	13-JAN-00	SH_CLERK	DGRANT	Grant
120	8000	18-JUL-96	ST_MAN	MWEISS	Weiss
121	8200	10-APR-97	ST_MAN	AFRIIPP	Fripp
122	7900	01-MAY-95	ST_MAN	PKAUFLIN	Kaufling
123	6500	10-OCT-97	ST_MAN	SVOLLMAN	Vollman
124	5800	16-NOV-99	ST_MAN	KMOURGOS	Mourgos
125	3200	16-JUL-97	ST_CLERK	JNAYER	Nayer
126	2700	28-SEP-98	ST_CLERK	IMIKKILI	Mikkilineni
...					
45 rows selected.					

On successful insertion of the information about the two new employees, you can query the NEW_EMPLOYEES table again.

EMPLOYEE_ID	SALARY	HIRE_DATE	JOB_ID	EMAIL	LAST_NAME
198	2600	21-JUN-99	SH_CLERK	DOCONNEL	OConnell
199	2600	13-JAN-00	SH_CLERK	DGRANT	Grant
120	8000	18-JUL-96	ST_MAN	MWEISS	Weiss
...					
800	1000	21-MAY-05	ST_CLERK	PKEVIN	Kevin
900	3000	12-MAY-04	ST_CLERK	JMATHEW	MATHEW
...					
47 rows selected.					

The last two rows, with employee IDs 800 and 900, are the newly added rows.

The complete code for the example in the slide is:

```
DECLARE
    insCtx DBMS_XMLSTORE.ctxType;
    rows NUMBER;
    xmlDoc CLOB := 
        '<ROWSET>
            <ROW num="1">
                <EMPLOYEE_ID>800</EMPLOYEE_ID>
                <SALARY>1000</SALARY>
                <DEPARTMENT_ID>50</DEPARTMENT_ID>
                <HIRE_DATE>21-May-2005</HIRE_DATE>
                <LAST_NAME>Kevin</LAST_NAME>
                <EMAIL>PKEVIN</EMAIL>
                <JOB_ID>ST_CLERK</JOB_ID>
            </ROW>
            <ROW num="2">
                <EMPLOYEE_ID>900</EMPLOYEE_ID>
                <SALARY>3000</SALARY>
                <DEPARTMENT_ID>50</DEPARTMENT_ID>
                <HIRE_DATE>12-MAY-2004</HIRE_DATE>
                <LAST_NAME>MATHEW</LAST_NAME>
                <EMAIL>JMATHEW</EMAIL>
            <JOB_ID>ST_CLERK</JOB_ID>
        </ROW>
    </ROWSET>';
BEGIN
    insCtx := DBMS_XMLSTORE.newContext('HR.NEW_EMPLOYEES');
    DBMS_XMLSTORE.clearUpdateColumnList(insCtx);
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'EMPLOYEE_ID');
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'SALARY');
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'HIRE_DATE');
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'DEPARTMENT_ID');
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'JOB_ID');
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'EMAIL');
    DBMS_XMLSTORE.setUpdateColumn(insCtx, 'LAST_NAME');
    rows := DBMS_XMLSTORE.insertXML(insCtx, xmlDoc);
    DBMS_OUTPUT.put_line(rows || ' rows inserted.');
    DBMS_XMLSTORE.closeContext(insCtx);
END;
/
```

Updating with DBMS_XMLSTORE

```

DECLARE
  updCtx DBMS_XMLSTORE.ctxType;
  ...
  <EMPLOYEE_ID>144</EMPLOYEE_ID>
    <FIRST_NAME>Anderson</FIRST_NAME>
  </ROW>
</ROWSET>';

BEGIN
  updCtx :=
    DBMS_XMLSTORE.newContext('HR.NEW_EMPLOYEES');
  DBMS_XMLSTORE.clearUpdateColumnList(updCtx);
  DBMS_XMLSTORE.setKeyColumn(updCtx, 'EMPLOYEE_ID'); 1
  rows := DBMS_XMLSTORE.updateXML(updCtx, xmlDoc); 2
  DBMS_XMLSTORE.closeContext(updCtx);
END;

```

ORACLE

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

You can use the DBMS_XMLSTORE PL/SQL package to update data in XML documents. When you perform an update operation by using DBMS_XMLSTORE, you need to use other procedures in the PL/SQL package to help the database understand the following:

- The columns that you are updating
- The keys that are the update keys

You must specify which rows to update by calling the setKeyColumn procedure once for each of the columns that are used collectively to identify the row. This set of key columns specifies the unique row to be updated. However, the columns that you use with setKeyColumn need not be the actual keys of the table. The only condition is that they should uniquely specify a row to be used with the calls to setKeyColumn.

The following query on the NEW_EMPLOYEES table shows the first name of the employee with ID 144 as Peter:

```

SELECT employee_id, first_name
FROM new_employees
WHERE employee_id = 144;

```

```
EMPLOYEE_ID FIRST_NAME
```

```
-----  
144 Peter
```

In the example, note the following:

1. The `employee_id` column is the key column. Because the `employee_id` column is a primary key for the `EMPLOYEES` table, a single call to `setKeyColumn` specifying the `employee_id` column is sufficient to identify a unique row for updating.
2. `updateXML` executes the `UPDATE` statement and returns the number of rows updated.

The complete example is as follows:

```
DECLARE  
    updCtx DBMS_XMLSTORE.ctxType;  
    rows NUMBER;  
    xmlDoc CLOB :=  
        '<ROWSET>  
            <ROW>  
                <EMPLOYEE_ID>144</EMPLOYEE_ID>  
                <FIRST_NAME>Anderson</FIRST_NAME>  
            </ROW>  
        </ROWSET>' ;  
  
BEGIN  
    updCtx := DBMS_XMLSTORE.newContext('HR.NEW_EMPLOYEES') ;  
    DBMS_XMLSTORE.clearUpdateColumnList(updCtx) ;  
    -- Specify that column employee_id is a "key"  
    -- to identify the row to update.  
    DBMS_XMLSTORE.setKeyColumn(updCtx, 'EMPLOYEE_ID') ;  
    rows := DBMS_XMLSTORE.updateXML(updCtx, xmlDoc) ;  
    DBMS_XMLSTORE.closeContext(updCtx) ;  
  
END;  
/
```

After the update, you can query to check whether the first name is modified:

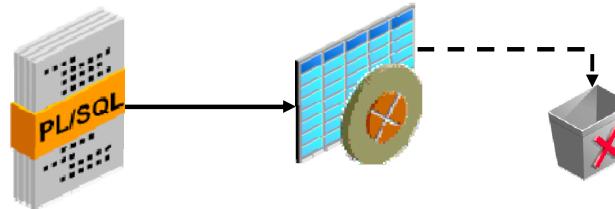
```
SELECT employee_id, first_name  
FROM new_employees  
WHERE employee_id = 144;
```

```
EMPLOYEE_ID FIRST_NAME
```

```
-----  
144 Anderson
```

Deleting with DBMS_XMLSTORE

```
DECLARE
  delCtx DBMS_XMLSTORE.ctxType;
...
<EMPLOYEE_ID>144</EMPLOYEE_ID>
  <DEPARTMENT_ID>50</DEPARTMENT_ID>
...
BEGIN
  delCtx := DBMS_XMLSTORE.newContext('HR.NEW_EMPLOYEES');
  DBMS_XMLSTORE.setKeyColumn(delCtx, 'EMPLOYEE_ID');
  rows := DBMS_XMLSTORE.deleteXML(delCtx, xmlDoc);
  DBMS_XMLSTORE.closeContext(delCtx);
END;
```



ORACLE

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

The DBMS_XMLSTORE.deleteXML PL/SQL function deletes the specified records from the XML document. With DBMS_XMLSTORE, deletions are treated similarly as updates. However, you must specify the (pseudo) key columns that identify the rows to be deleted.

The example in the slide shows how to delete the record for the employee with ID 144 by using DBMS_XMLSTORE. The employee_id column is specified as the key column that identifies the row to be deleted.

Before you perform the delete operation, you can query the table:

```
SELECT employee_id
FROM new_employees
WHERE employee_id = 144;
```

```
EMPLOYEE_ID
-----
144
```

After you perform the delete operation, you can query the table to check the result:

```
SELECT employee_id  
FROM new_employees  
WHERE employee_id = 144;
```

no rows selected.

The complete example is as follows:

```
DECLARE  
    delCtx DBMS_XMLSTORE.ctxType;  
    rows NUMBER;  
    xmlDoc CLOB :=  
        '<ROWSET>  
            <ROW>  
                <EMPLOYEE_ID>144</EMPLOYEE_ID>  
                    <DEPARTMENT_ID>50</DEPARTMENT_ID>  
                </ROW>  
            </ROWSET>' ;  
BEGIN  
    delCtx := DBMS_XMLSTORE.newContext('HR.NEW_EMPLOYEES') ;  
    DBMS_XMLSTORE.setKeyColumn(delCtx, 'EMPLOYEE_ID') ;  
    rows := DBMS_XMLSTORE.deleteXML(delCtx, xmlDoc) ;  
    DBMS_XMLSTORE.closeContext(delCtx) ;  
END ;  
/
```

Lesson Agenda

- PL/SQL APIs for XMLType
- Using DBMS_XMLSTORE
- Document Object Model (DOM)
- Using DBMS_XMLDOM to:
 - Create a DOM document
 - Modify a DOM document
 - Add nodes to a DOM document
- Using DBMS_XMLPARSER
- Using DBMS_XSLPROCESSOR



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

Document Object Model of XML

XML data is structured in the form of a tree. Parsing an XML document into an in-memory, tree-based DOM model makes it possible to operate on the tree structure of the XML data to:

- Traverse the tree programmatically
- Directly access the components of the document
- Rearrange or remove the components of the document
- Directly access data



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

XML documents are hierarchical in structure, thus making it easy to store them as trees. After you parse an XML document into a tree structure, you can operate on the tree. This yields a great deal of flexibility in dealing with the documents. You can access the components of a document in any order, rearrange them, and add or remove them.

This is especially appropriate for applications in which the flow of processing is based on external logic, as opposed to the order and occurrence of elements within the XML document. Storing the document as a tree enables direct access to its data and structure, instead of having the processing governed by when and where the tags and elements occur.

However, tree-based methodologies do have some drawbacks. They require parsing the entire XML document and creating the tree data structure before the processing and business logic can take place.

W3C Document Object Model Recommendation: Overview

The W3C DOM recommendation is a specification for APIs that access the structure of XML documents. According to W3C, the DOM should:

- Provide a language- and platform-neutral object model for XML documents
- Facilitate the writing of applications that dynamically delete, add, and edit the content, attributes, and style of XML documents



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

The Document Object Model (DOM) recommended by the World Wide Web Consortium (W3C) is a universal specification for APIs that access the structure of XML documents. It was originally developed to formalize dynamic HTML, which allows animation, interaction, and dynamic updating of web pages. The DOM provides a language- and platform-neutral object model for web pages and XML document structures. A DOM is an in-memory, tree-based object representation of an XML document that enables programmatic access to the elements and attributes of the document. The DOM facilitates the writing of applications that dynamically delete, add, and edit the content, attributes, and style of XML documents.

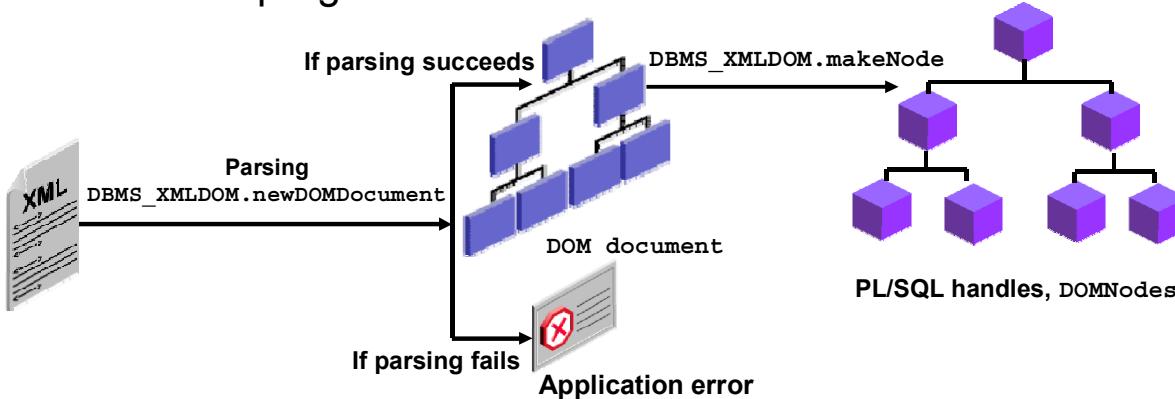
DOM Level 1.0: Is the first formal level of DOM specifications, which was completed in October 1998. Level 1.0 defines support for XML 1.0 and HTML

DOM Level 2.0: Was completed in November 2000. Level 2.0 extends Level 1.0 with support for XML namespaces, cascading style sheets (CSS), user interface events, and tree manipulation events. CSS are used for adding style (fonts, colors, spacing, and so on) to web documents

Processing XML Structures with DBMS_XMLDOM

DBMS_XMLDOM:

- Parses an XML document into a tree-based DOM document instance
- Converts various DOM types (document, element, and attributes) into PL/SQL handles (DOMNodes)
- Enables programmatic access to DOMNodes



ORACLE

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

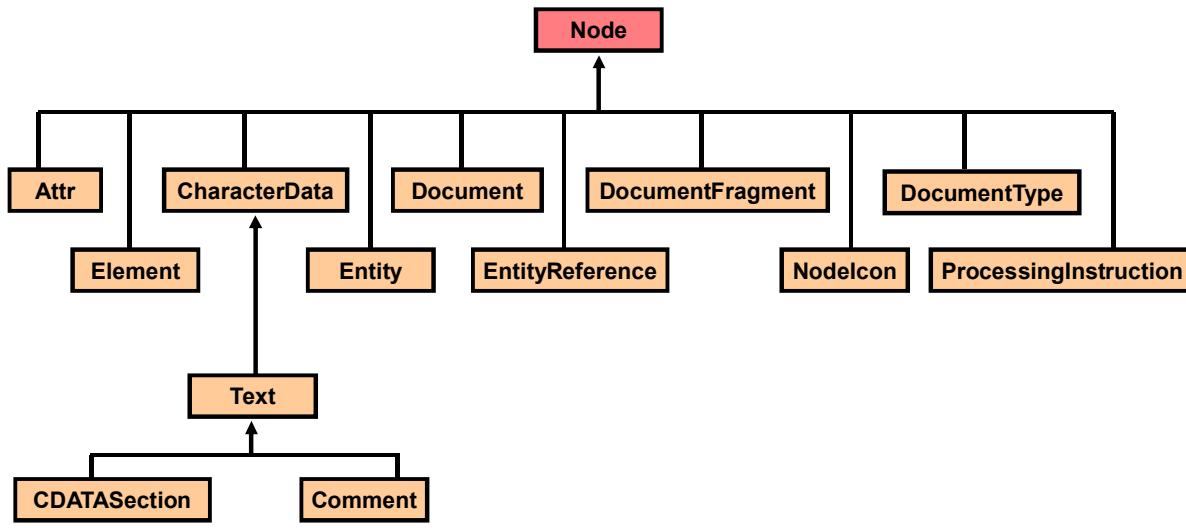
Oracle XML DB extends the Oracle Database XML development platform beyond SQL support for storage and retrieval of XML data. It enables you to operate on `XMLType` instances by using the DOM in PL/SQL, Java, and C.

Starting with Oracle Database 10g, the PL/SQL API for `XMLType` implements DOM Level 1.0 and Level 2.0, and is fully integrated in the database through extensions to the `DBMS_XMLDOM` `XMLType` API. The Oracle PL/SQL DOM API supports both XML schema-based documents and non-schema-based documents.

The `DBMS_XMLDOM` package facilitates creation of DOM documents. Because PL/SQL does not directly support the node-based hierarchical structure of an XML document, it is implemented through direct invocation of the `makeNode` function. Calling `makeNode` on various DOM types (document, element, attributes, and so on) converts these types into their respective PL/SQL handles, which are called `DOMNodes`. The appropriate functions or procedures that accept `DOMNodes` can then be called to operate on these types.

DOM Node Types: Inheritance

Inheritance structure for DOM node types:

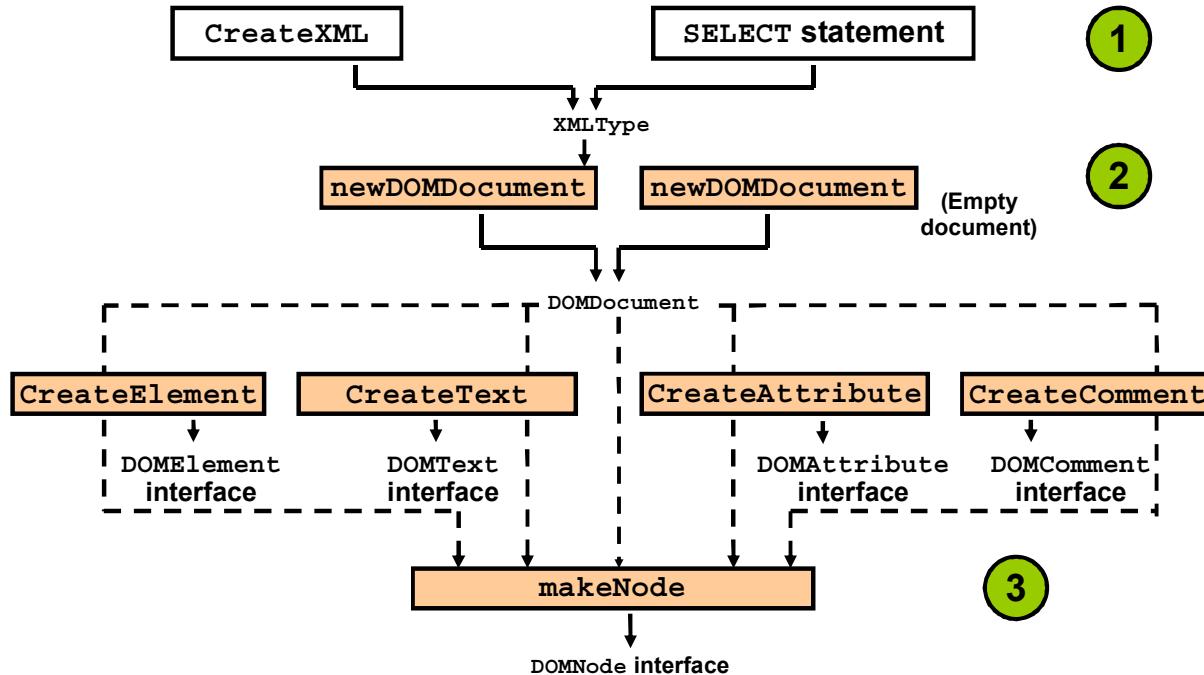


ORACLE

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

The DOM defines an inheritance hierarchy. For example, Document, Element, and Attr are defined to be subtypes of Node. Thus, a method defined in the Node interface should be available in these subtypes as well. For every DOM interface, there is a corresponding DBMS_XMLDOM type, such as DOMNODE, DOMATTR, DOMCDATASECTION, DOMDOCUMENT, DOMELEMENT, DOMDOCUMENTFRAGMENT, DOMENTITY, and so on.

Using DBMS_XMLDOM to Create DOM Document Handles



ORACLE

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

To create DOM document handles, perform the following:

1. Fetch or construct an `XMLType` instance.
2. Create a `DOMDocument` instance by using the `newDOMDocument` procedure, which processes the `XMLType` instance or the empty document.
3. Create a handle to this DOM document by using the `makeNode()` procedure.
 - a. You can use DOM API methods such as `createElement()`, `createText()`, `createAttribute()`, and `createComment()` to traverse and extend the DOM tree.
 - b. The results of methods, such as `DOMELEMENT` and `DOMTEXT`, can also be passed to `makeNode()` to obtain the `DOMNode` interface.

Creating a DOM Document

Example of DOM document creation:

```
DECLARE
    xmlTableVar XMLType;
    DOMDocVar DBMS_XMLDOM.DOMDocument;
    nodeVar DBMS_XMLDOM.DOMNode;
    buf VARCHAR2(2000);

BEGIN
    xmlTableVar :=
        XMLType('<PERSON><NAME>KING</NAME></PERSON>');
    -- Create DOMDocument handle
    DOMDocVar := DBMS_XMLDOM.newDOMDocument(xmlTableVar);
    nodeVar := DBMS_XMLDOM.makeNode(DOMDocVar);
    DBMS_XMLDOM.writeToBuffer(nodeVar, buf);
    DBMS_OUTPUT.put_line('Contents of the XML
Doc:' || buf);
END;
/
```

ORACLE

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

The example in the slide creates a DOM document for an XMLType variable by calling the DBMS_XMLDOM.newDOMDocument() procedure and passing the XML document as a parameter. It then calls DBMS_XMLDOM.makeNode() to create a handle to access the DOM document that is passed as a parameter. PL/SQL can now use this handle to access and modify the XMLType data.

The slide displays one way of accessing the contents of DOMNode, that is, to write the contents of the node to a buffer by using the writeToBuffer() procedure.

Output:

```
Contents of the XML Doc:<PERSON>
                                <NAME>KING</NAME>
                                </PERSON>
```

Accessing and Manipulating a DOM Document

```

DECLARE
    ...
BEGIN
    ...
-- Access element
    docelem := DBMS_XMLDOM.getDocumentElement(doc); 1
    nodelist := DBMS_XMLDOM.getElementsByTagName(docelem,
                                                'NAME'); 2
    node := DBMS_XMLDOM.item(nodelist, 0);
    childnode := DBMS_XMLDOM.getFirstChild(node); 3
-- Manipulate element
    DBMS_XMLDOM.setNodeValue(childnode, 'RICK'); 4
    DBMS_XMLDOM.writeToBuffer(ndoc, buf);
    DBMS_OUTPUT.put_line('After:' || buf);
    DBMS_XMLDOM.freeDocument(doc);
END;
/

```

ORACLE

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

Functions and Procedures

1. `getDocumentElement()`
 - The `doc` variable is the `DOMDocument` holding the parsed XML tree.
 - This function returns a `DOMELEMENT` corresponding to the root element of the `DOMDocument`.
2. `getElementsByTagName()`
 - The `name` variable is the tag name of the element that needs to be retrieved.
 - The `docelem` variable is the `DOMELEMENT` holding the parsed XML tree.
 - This function returns a `DOMNodeList` of elements that match the given tag name.
3. `item()`
 - The `nodelist` variable is a `DOMNodeList`.
 - The second parameter is a numeric value specifying the index of the item to be retrieved.
If `index` is greater than or equal to the number of nodes, it returns `NULL`.
 - This function returns a `DOMNode` of elements that match the given index.

4. `getFirstChild()`
 - node is a DOMNODE of which this function retrieves the first child.
 - This function returns a DOMNODE; if there is no such node, it returns NULL.
 - To retrieve the last child of a node, a similar `getLastChild()` function is used.
5. `setNodeValue`
 - The `childnode` variable is a DOMNODE of which this procedure changes the value.
 - RICK is the new value assigned to `childnode`.

Programmatic Use of DBMS_XMLDOM Subprograms

The following program creates a DOM document handle for an XML document, accesses the document, and later manipulates it:

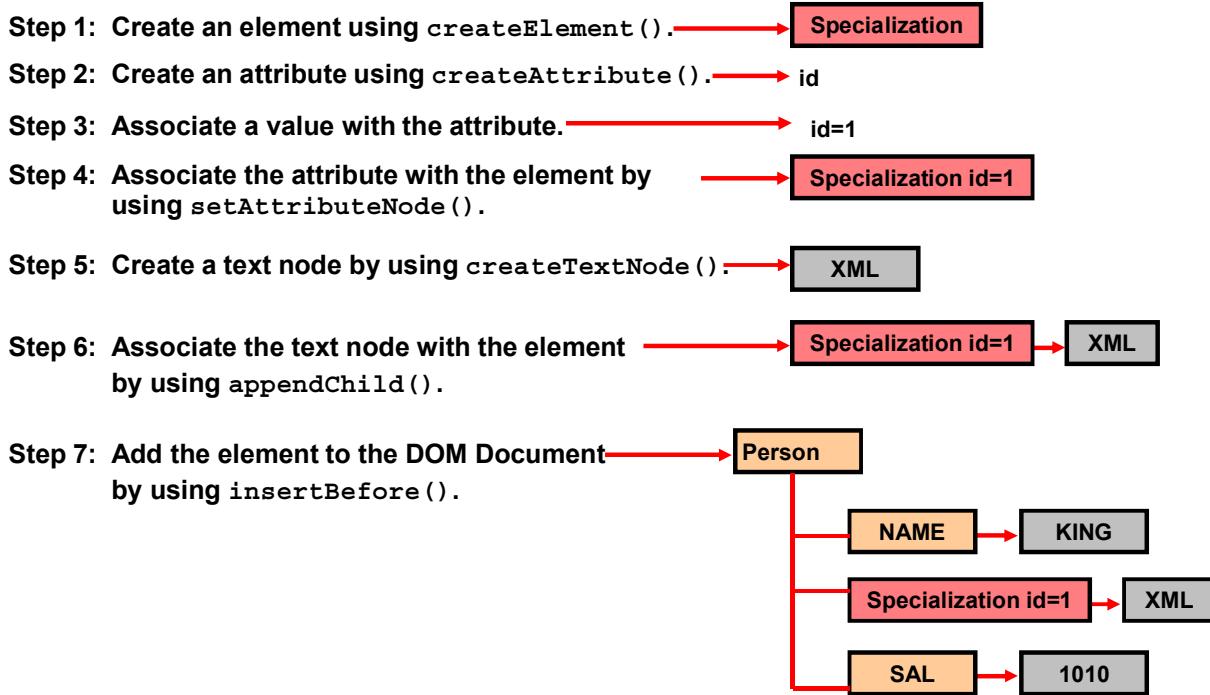
```

DECLARE
    var XMLType;
        doc DBMS_XMLDOM.DOMDocument;
        ndoc DBMS_XMLDOM.DOMNode;
        docelem DBMS_XMLDOM.DOMElement;
        node DBMS_XMLDOM.DOMNode;
        childnode DBMS_XMLDOM.DOMNode;
        nodelist DBMS_XMLDOM.DOMNodelist;
        buf VARCHAR2(2000);

BEGIN
    var := XMLType ('<PERSON><NAME>KING</NAME></PERSON>');
        -- Create DOMDocument handle
    doc := DBMS_XMLDOM.newDOMDocument (var);
    ndoc := DBMS_XMLDOM.makeNode (doc);
    DBMS_XMLDOM.writeToBuffer (ndoc, buf);
    DBMS_OUTPUT.put_line ('Before:' || buf);
        -- Access element
    docelem := DBMS_XMLDOM.getDocumentElement (doc);
    nodelist :=
        DBMS_XMLDOM.getElementsByTagName (docelem,
    'NAME');
    node := DBMS_XMLDOM.item (nodelist, 0);
    childnode := DBMS_XMLDOM.getFirstChild (node);
        -- Manipulate element
    DBMS_XMLDOM.setNodeValue (childnode, 'Rick');
    DBMS_XMLDOM.writeToBuffer (ndoc, buf);
    DBMS_OUTPUT.put_line ('After:' || buf);
    DBMS_XMLDOM.freeDocument (doc);
END;
/

```

Adding Nodes to a DOM Document



ORACLE

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

The slide shows the stepwise insertion of an element into a DOM document. Steps 2 through 4 show the addition of an attribute to an element. Steps 5 and 6 show the addition of text nodes to the element. The newly created element is added to the DOM document in step 7.

Adding Nodes to a DOM Document

```

CREATE OR REPLACE PROCEDURE ADDNODE IS
  ...
  BEGIN
    ...
    n_elem :=DBMS_XMLDOM.createElement ( doc, 'Specialization'); 1

    v_attr :=DBMS_XMLDOM.createAttribute( doc, 'id');
    n_attr :=DBMS_XMLDOM.MakeNode(v_attr);
    DBMS_XMLDOM.setNodeValue ( n_attr, '1');
    v_attr :=DBMS_XMLDOM.setAttributeNode (n_elem, v_attr); 2

    node := DBMS_XMLDOM.MakeNode(n_elem);
    n_txt :=DBMS_XMLDOM.MakeNode(DBMS_XMLDOM.createTextNode(
      doc, 'XML') );
    n_txt :=DBMS_XMLDOM.appendChild( node, n_txt); 3

    node := DBMS_XMLDOM.insertBefore(ndoc, node, childnode); 4
    ...
    DBMS_XMLDOM.writeToFile(ndoc, v_dir|| '\abc.txt'); 5
    ...
  END;

```

ORACLE

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

1. Creating the element (function)

`createElement ()`: Creates and returns `DOMELEMENT`. A DOM document named `doc`, holding the parsed XML tree to which the element is being added, is passed as the first parameter. The name of `DOMELEMENT` is passed as the second parameter.

2. Creating the attribute and attaching it to a specific node (functions and procedures)

`createAttribute ()`: Creates and returns `DOMAttr` having a name equal to the second parameter. The DOM document named `doc` is passed as the first parameter.

`makeNode ()`: Returns `DOMNode` created by casting `DOMAttr`

Procedure `setNodeValue ()`: Sets `DOMNode` '`n_attr`' (which is passed as the first parameter) to the value passed as the second parameter

`setAttributeNode ()`: Attaches `DOMAttr` '`_attr`' (which is passed as the second parameter) to `DOMNode` passed as the first parameter

3. Creating the text node and attaching it to a specific node (functions)

`createTextNode()`: Creates and returns DOMText having a varchar value equal to the second parameter. The DOM document named `doc` is passed as the first parameter. Later, DOMText is cast to DOMNode '`n_txt`' by using `makeNode()`.

`appendChild()`: Attaches DOMNode '`n_txt`' (which is passed as the second parameter) to DOMNode passed as the first parameter

4. Attaching the node to a specific location (function)

`insertBefore()`: Inserts DOMNode passed as the second parameter, `node`, before DOMNode passed as the third parameter, `childnode`. The DOM document named `doc` is passed as the first parameter.

5. Writing DOMNode to a file (procedure)

`writeToFile()`: Writes the XML node passed as the first parameter to the file specified by the URL in the second parameter. However, as a prerequisite for this command, the path of the directory must be specified by a sysdba user by using the following command. (The database needs to be restarted.)

```
alter system set UTL_FILE_DIR= 'D:\labs\xml_dir' scope=spfile;
```

Complete Code

```
CREATE OR REPLACE PROCEDURE ADDNODE IS
  var XMLType;
  doc DBMS_XMLDOM.DOMDocument;
  ndoc DBMS_XMLDOM.DOMNode;
  docelem DBMS_XMLDOM.DOMElement;
  node DBMS_XMLDOM.DOMNode;
  childnode DBMS_XMLDOM.DOMNode;
  buf VARCHAR2(2000);
  v_dir VARCHAR2(2000);
  n_elem DBMS_XMLDOM.DOMElement;
  v_attr DBMS_XMLDOM.DOMAttr;
  n_attr DBMS_XMLDOM.DOMNode;
  n_txt DBMS_XMLDOM.DOMNode;
```

```
BEGIN
    v_dir := 'D:\labs\xml_dir';
    var := XMLType ('<PERSON>
        <NAME>KING</NAME><SAL>1010</SAL></PERSON>');
    doc := DBMS_XMLDOM.newDOMDocument (var);
    ndoc :=
        DBMS_XMLDOM.makeNode (DBMS_XMLDOM.getDocumentElement (doc));
    childnode := DBMS_XMLDOM.getLastChild(ndoc);
        --creating new elements
    n_elem :=DBMS_XMLDOM.createElement (doc, 'Specialization');
    v_attr :=DBMS_XMLDOM.createAttribute( doc, 'id');
    n_attr :=DBMS_XMLDOM.MakeNode(v_attr);
    DBMS_XMLDOM.setNodeValue ( n_attr, '1');

    v_attr :=DBMS_XMLDOM.setAttributeNode (n_elem, v_attr);
    node :=DBMS_XMLDOM.MakeNode(n_elem);
    n_txt :=DBMS_XMLDOM.MakeNode (DBMS_XMLDOM.createTextNode (doc,
        'XML'));
    n_txt :=DBMS_XMLDOM.appendChild( node, n_txt);
        --Adding the new element
    node := DBMS_XMLDOM.insertBefore(ndoc, node, childnode);
    DBMS_XMLDOM.writeToFile(ndoc, v_dir||'\abc.txt');
    DBMS_XMLDOM.freeDocument(doc);
END;
/
EXECUTE ADDNODE;
```

Output: The abc.txt file is created in D:\labs\xml_dir with the following content:

```
<PERSON> <NAME>KING</NAME>
<Specialization id='1'>XML</Specialization>
<SAL>1010</SAL></PERSON>
```

Lesson Agenda

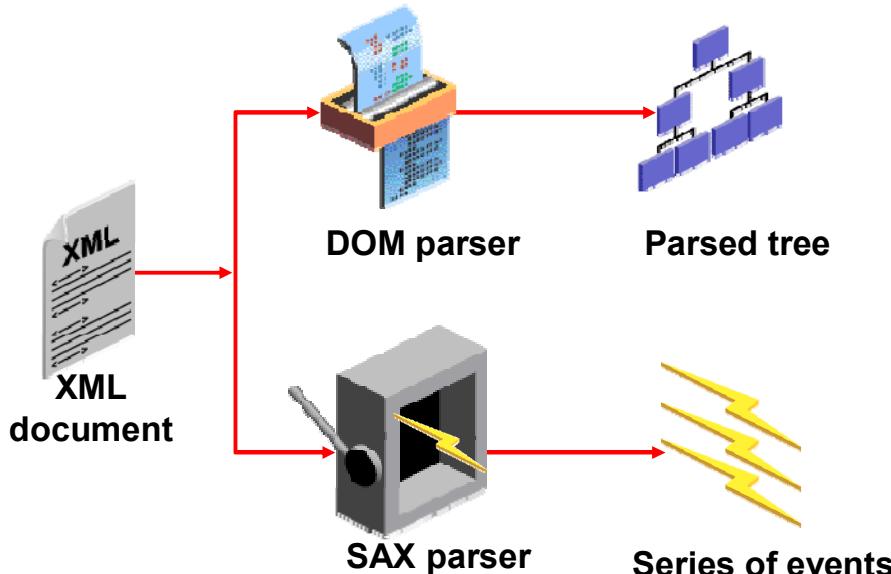
- PL/SQL APIs for XMLType
- Using DBMS_XMLSTORE
- Document Object Model (DOM)
- Using DBMS_XMLDOM to:
 - Create a DOM document
 - Modify a DOM document
 - Add nodes to a DOM document
- Using DBMS_XMLPARSER
- Using DBMS_XSLPROCESSOR



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

XML Parsers

XML parsers are the components that provide programmatic access to XML documents.



ORACLE

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

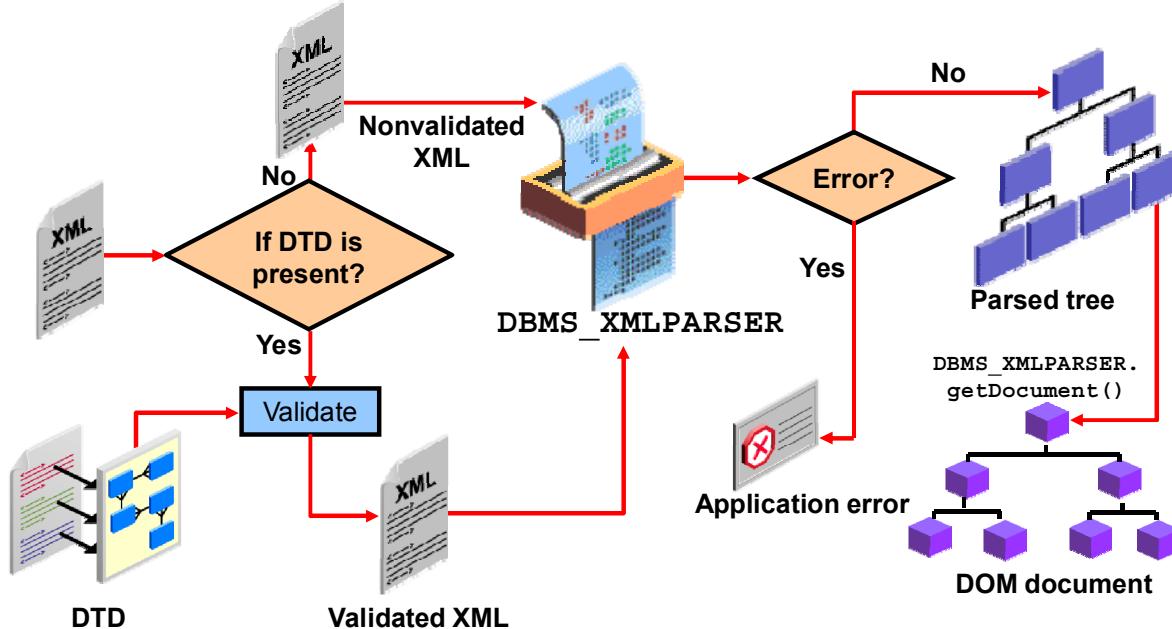
The foundation component of any XML development is the XML parser. The parser provides programmatic access to XML documents that you can use in many ways in an application. Oracle parsers support two industry-standard specifications for XML document access: DOM and SAX. The parsers support W3C-standard DOM 2.0 CORE specifications. They also support SAX 2.0, which adds namespace support. The Oracle XML parser is implemented in C, C++, PL/SQL, and Java.

Difference Between DOM and SAX

- **Document Object Model (DOM):** The DOM is the primary, generic, *tree-based* API for XML. It works by creating objects that have properties and child objects. Objects in a DOM tree are referenced either by moving down the object hierarchy or by explicitly giving an XML element an ID attribute.
- **Simple API for XML (SAX):** SAX is the primary, generic, *event-based* programming interface between an XML parser and an XML application. Instead of building a complete representation of the document, a SAX parser fires off a series of events as it reads the document from beginning to end.

This slide shows the original XML document and the Extensible Stylesheet Language (XSL) stylesheet going through the XML parser. The XML document passes through the DOM parser and sends the parsed XSL commands and the parsed XML to the XSL Transformation (XSLT) processor, which uses the XSL stylesheet to transform the original XML document into the transformed XML document.

PL/SQL Parser API for XMLType (DBMS_XMLPARSER)



ORACLE

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

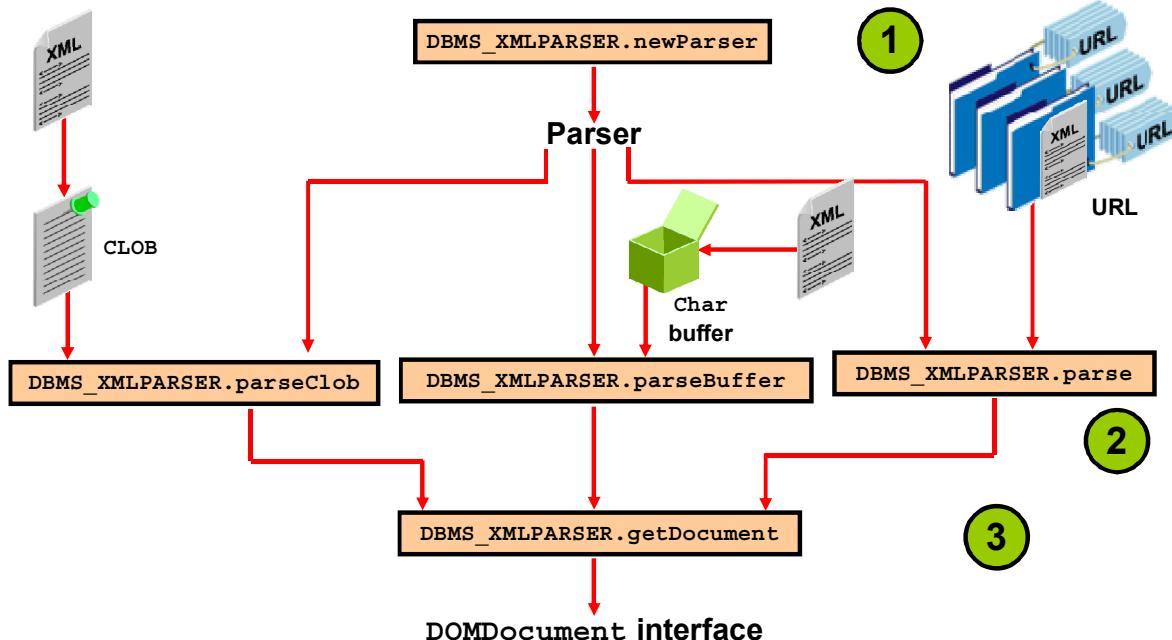
By using the PL/SQL Parser API for XMLType (DBMS_XMLPARSER), you can access the contents and structure of XML documents. DBMS_XMLPARSER builds a result tree that can be accessed by PL/SQL APIs. If parsing fails, an error is raised.

A software module called an XML parser is used to read XML documents and provide access to their content and structure. The PL/SQL implementation of the XML parser follows the W3C XML specification REC-xml-19980210 and includes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

To parse an XML document in Oracle XML DB and to obtain a `DOMDocument` object, you can use either the `DBMS_XMLPARSER` package or the `XMLType` constructors followed by the `DBMS_XMLDOM.newDOMDocument` function.

The default behavior for `DBMS_XMLPARSER` is to build a parse tree that can be accessed by DOM APIs, validate it if a document type definition (DTD) is found (otherwise, it is nonvalidating), and record errors if an error log is specified. If parsing fails, an application error is raised.

Obtaining a DOM Document Interface by Using DBMS_XMLPARSER



ORACLE

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

To use the PL/SQL Parser API for XMLType (DBMS_XMLPARSER):

1. Construct a parser instance by using the DBMS_XMLPARSER.newParser method
2. Parse XML documents. An error is raised if the input is not a valid XML document.
 - a. Use the DBMS_XMLPARSER.parseBuffer method to parse the XML document stored in the given buffer.
 - b. Use the DBMS_XMLPARSER.parseClob method to parse the XML document stored in the given character large object (CLOB).
 - c. Use the DBMS_XMLPARSER.parse method to parse the XML document stored in the given Uniform Resource Identifier (URI). DBMS_XMLPARSER.setBaseDir should be executed to set the directory path of the XML file before parsing the XML data that is stored in the URI.

To use DBMS_XMLPARSER.parse, you must connect as sys and set UTL_FILE_DIR by using the following command. After you set this command, you must restart the database:

```
Alter system set UTL_FILE_DIR = 'D:\labs\xml_dir'
scope=spfile
```

3. Call the DBMS_XMLPARSER.getDocument method on the parser to obtain the DOMDocument interface that was created during the parsing of the XML document.

Parsing an XML Document Without DTD Reference by Using DBMS_XMLPARSER

Parsing XML stored in a buffer (example):

```
DECLARE
    indoc VARCHAR2(2000);
    indomdoc DBMS_XMLDOM.DOMDocument;
    innode DBMS_XMLDOM.DOMNode;
    myparser DBMS_XMLPARSER.parser;
    buf VARCHAR2(2000);
BEGIN
    indoc := '<emp><name>KING</name></emp>';
    myParser := DBMS_XMLPARSER.newParser;
    DBMS_XMLPARSER.parseBuffer(myParser, indoc);
    indomdoc := DBMS_XMLPARSER.getDocument(myParser);
    innode := DBMS_XMLDOM.makeNode(indomdoc);
    DBMS_XMLDOM.writeToBuffer(innode, buf);
    DBMS_OUTPUT.put_line(buf);
    DBMS_XMLDOM.freeDocument(indomdoc);
    DBMS_XMLPARSER.freeParser(myParser);
END;
/
```

ORACLE

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

The example in the slide parses a simple XML document. It creates an XML parser (instance of DBMS_XMLPARSER.parser) and uses it to parse the XML document (text) in the indoc variable. Parsing creates a DOM document that is retrieved from the parser by using DBMS_XMLPARSER.getDocument(). A DOM node is created that contains the entire document, and the node is then printed. After freeing (destroying) the DOM document, the parser instance is freed by using DBMS_XMLPARSER.freeParser().

Output

```
<emp>
  <name>KING</name>
</emp>
```

Parsing an XML Document with DTD Reference by Using DBMS_XMLPARSER

```

DECLARE
    res boolean;
    dtd clob := '<!ELEMENT projects (project+)>
    ...';
BEGIN
    res := dbms_xdb.createResource('/projects.dtd',dtd);
    commit;
END;

```

1

```

<?xml version="1.0"?>
<!DOCTYPE projects SYSTEM "projects.dtd">
<projects>
    ...
</projects>

```

2

```

CREATE OR REPLACE PROCEDURE USEPARSER IS
    . . .
BEGIN
    . . .
        DBMS_XMLPARSER.setBaseDIR(myparser, v_dir);
        DBMS_XMLPARSER.parse(myParser,'projects.xml');
    . . .
End;

```

3

ORACLE

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

The example in the slide uses `DBMS_XMLPARSER.parse()` to parse an XML document based on a document type definition (DTD).

Step 1: Create a DTD and store it in the root (/) of Oracle XML DB Repository.

Note: To create a Repository resource under root (/), you need to have the XDBADMIN privilege.

```
DECLARE
```

```

    res boolean;
    dtd clob := '<!ELEMENT projects (project+)>
        <!ELEMENT project (goal, team_member+)>
        <!ATTLIST project project_id ID #REQUIRED>
        <!ELEMENT goal (#PCDATA)>
        <!ELEMENT team_member EMPTY>
        <!ATTLIST team_member person CDATA #REQUIRED>';

```

```
BEGIN
    res := dbms_xdb.createResource('/projects.dtd',dtd);
    commit;
END;
```

Step 2: Create an XML file that references the DTD by using the DOCTYPE tag. If DOCTYPE of a DTD is SYSTEM rather than PUBLIC, Oracle XML DB treats the path as being relative to the root of Oracle XML DB Repository.

```
<?xml version="1.0"?>
<!DOCTYPE projects SYSTEM "projects.dtd">
<projects>
    <project project_id="p1">
        <goal>Develop Strategic Plan</goal>
        <team_member person="ss078-05-1120"/>
        <team_member person="ss987-65-4320"/>
    </project>
    <project project_id="p2">
        <goal>Deploy Linux</goal>
        <team_member person="ss078-05-1120"/>
        <team_member person="ss9876-12-3456"/>
    </project>
</projects>
```

Step 3: Create a procedure to parse the XML file. The useparser() procedure uses DBMS_XMLPARSER.setBaseDIR() to set D:\labs\xml_dir as the base directory used to resolve relative URLs. The myparser parser (an instance of DBMS_XMLPARSER.parser) then parses the XML file projects.xml after validating it on the basis of the DTD /projects.dtd.

```
CREATE OR REPLACE PROCEDURE useparser
IS
    indoc VARCHAR2(2000);
    myparser DBMS_XMLPARSER.parser;
    xmldoc DBMS_XMLDOM.DOMDocument;
    v_dir varchar2(30);
    ndoc DBMS_XMLDOM.DOMNode;
BEGIN
```

```
v_dir := 'D:\labs\xml_dir';
myParser := DBMS_XMLPARSER.newParser;
DBMS_XMLPARSER.setBaseDIR(myparser, v_dir);
DBMS_XMLPARSER.parse(myParser, 'projects.xml');
xmldoc := DBMS_XMLPARSER.getDocument(myParser);
ndoc := DBMS_XMLDOM.makeNode(xmldoc);
DBMS_XMLDOM.writeToFile(xmldoc, v_dir||'/DEF.txt');
DBMS_XMLPARSER.freeParser(myParser);
DBMS_XMLDOM.freeDocument(xmldoc);

END;
/
show errors
EXECUTE useparser
```

DBMS_XMLPARSER Subprograms

Output: The DEF.txt file is created in D:\labs\xml_dir with the following content:

```
<?xml version="1.0"?>
<!DOCTYPE projects[<!ELEMENT projects (project+)>
    <!ELEMENT project (goal, team_member+)>
    <!ATTLIST project project_id ID #REQUIRED>
    <!ELEMENT goal (#PCDATA)><!ELEMENT team_member EMPTY>
    <!ATTLIST team_member person CDATA #REQUIRED>] >
<projects>
    <project project_id="p1">
        <goal>Develop Strategic Plan</goal>
        <team_member person="ss078-05-1120"/>
        <team_member person="ss987-65-4320"/>
    </project>
    <project project_id="p2">
        <goal>Deploy Linux</goal>
        <team_member person="ss078-05-1120"/>
        <team_member person="ss9876-12-3456"/>
    </project>
</projects>
```

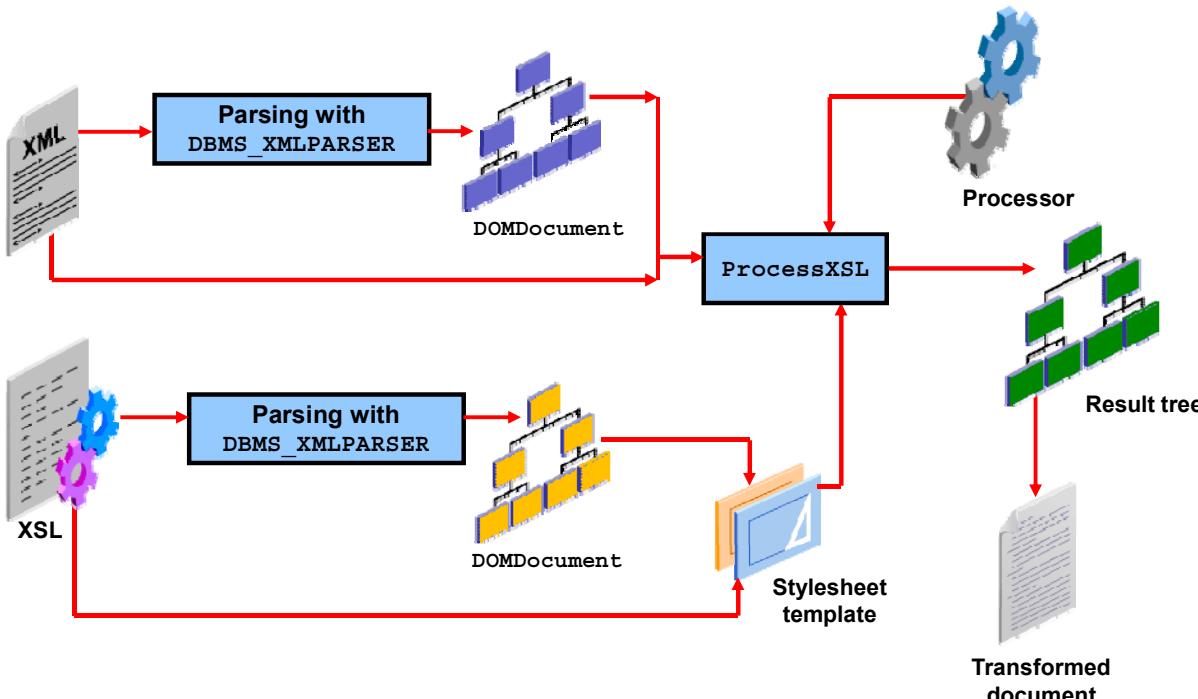
Lesson Agenda

- PL/SQL APIs for XMLType
- Using DBMS_XMLSTORE
- Document Object Model (DOM)
- Using DBMS_XMLDOM to:
 - Create a DOM document
 - Modify a DOM document
 - Add nodes to a DOM document
- Using DBMS_XMLPARSER
- Using DBMS_XSLPROCESSOR



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

PL/SQL XSLT Processor for XMLType (DBMS_XSLPROCESSOR)



ORACLE

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

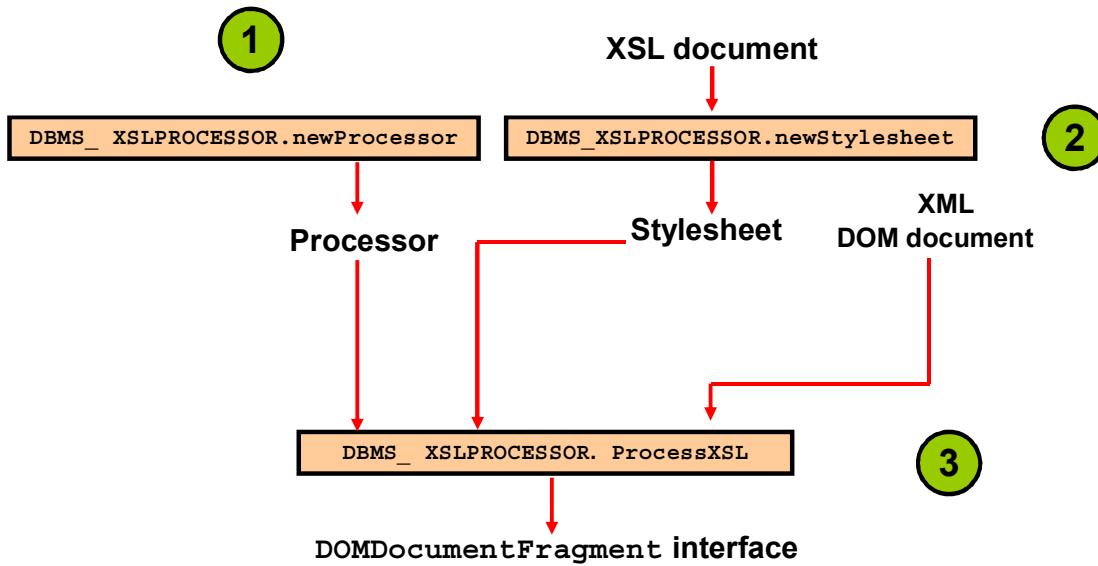
Oracle XML DB supports Extensible Stylesheet Language Transformation (XSLT). One of the important uses of XSLT is to convert an XML document to HTML so that it can be displayed on a web browser. With this API, an XML document can be transformed into another XML document or it can be converted into other supported formats, such as HTML and PDF.

To perform the transformations, patterns in the XML source tree are compared to certain rules called *templates* that are defined in the XSL stylesheet. An XSL stylesheet comprises transformations that are expressed in the XSLT language.

PL/SQL DBMS_XSLPROCESSOR follows the W3C XSLT final recommendation (REC-xslt-19991116), which states the action required of an XSL processor in terms of how it must read the XSLT stylesheets and the transformations it must achieve.

The PL/SQL DBMS_XSLPROCESSOR package makes available the types and methods of the PL/SQL XSLT Processor API. The methods in this package use two PL/SQL data types that are specific to the XSL processor implementation: PROCESSOR and STYLESTHEET.

Transforming an XML Document by Using DBMS_XSLPROCESSOR



ORACLE

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

To transform an XML document, perform the following steps:

1. Construct an XSLT processor by using `DBMS_XSLPROCESSOR.newProcessor()`.
2. Use `DBMS_XSLPROCESSOR.newStylesheet()` to build a `STYLESSHEET` object, from either a DOM document created from the XSL or by passing the URL of the XSL file.
3. Use `DBMS_XSLPROCESSOR.ProcessXSL()` to transform an XML document (source tree) by using the `PROCESSOR` (instance of `DBMS_XSLPROCESSOR.processor`) and `STYLESSHEET` object. The function returns the resultant `DOMDocumentFragment` (result tree).

Using DBMS_XSLPROCESSOR to Transform an XML Document: Example

```
BEGIN
...
--Create a stylesheet from the XSL document
  xsl := DBMS_XSLPROCESSOR.newStyleSheet(xsldoc, '');
--Create a new processor
  proc := DBMS_XSLPROCESSOR.newProcessor;
--Apply stylesheet to DOM document
  outdomdocfrag := DBMS_XSLPROCESSOR.processXSL(proc,
                                                 xsl, xmldoc);
...
END;
```



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

The example in the slide shows part of a PL/SQL program that uses DBMS_XSLPROCESSOR to apply a stylesheet to an XML DOM document. The code first generates a stylesheet for the XSL document stored in the xsldoc variable by using

DBMS_XSLPROCESSOR.newStyleSheet.

Next, this template tree is applied to the XML document xmldoc by using the DBMS_XSLPROCESSOR.processXSL function. A processor that is obtained by using DBMS_XSLPROCESSOR.newProcessor is also passed in this method.

Programmatic Use of DBMS_XMLPROCESSOR Subprograms: Example

```
CREATE TABLE emp_resumes(id number(4), resume XMLType);
INSERT INTO emp_resumes VALUES (10,
  XMLType('<?xml version="1.0"?>
<RESUME>
  <FULL_NAME>Steven King</FULL_NAME>
  <PHONE>20000</PHONE>
  <JOB_ID>AD_PRES</JOB_ID>
  <LOCATION>Boston</LOCATION>
</RESUME>'));
```

```
CREATE TABLE stylesheets(id number(4), xsld XMLType);
INSERT INTO STYLESHEETS VALUES (10,
    XMLType('<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="utf-8"/>
<!-- alphabetizes an xml tree -->
<xsl:template match="*">
    <xsl:copy>
        <xsl:apply-templates select="*|text()">
            <xsl:sort select="name(.)"
                data-type="text" order="ascending"/>
        </xsl:apply-templates>
    </xsl:copy>
</xsl:template>
<xsl:template match="text()">
    <xsl:value-of select="normalize-space(.)"/>
</xsl:template>
</xsl:stylesheet>'))
```

The following program creates a DOM document fragment for the XML document that has been transformed by the preceding XSL document:

```
DECLARE
    xmldoc Varchar2(2000);
    xsldoc VARCHAR2(2000);
    xsl DBMS_XSLPROCESSOR.stylesheet;
    outdomdocf DBMS_XMLDOM.DOMDocumentFragment;
    outnode DBMS_XMLDOM.DOMNode;
    proc DBMS_XSLPROCESSOR.processor;
    buf VARCHAR2(2000);
```

```
BEGIN
    select e.resume.getstringval() into xmldoc from
        emp_resumes e where id=10;
    select s.xsld.getstringval() into xsldoc from
        stylesheets s where id=1;
    xsl := DBMS_XSLPROCESSOR.newStyleSheet(xsldoc, '');
    proc := DBMS_XSLPROCESSOR.newProcessor;
    --apply stylesheet to DOM document
    outdomdocf := DBMS_XSLPROCESSOR.processXSL(proc, xsl,
        xmldoc);
    outnode := DBMS_XMLDOM.makeNode(outdomdocf);
    -- PL/SQL DOM API for XMLType can be used here
    DBMS_XMLDOM.writeToBuffer(outnode, buf);
    DBMS_OUTPUT.put_line(buf);
    DBMS_XMLDOM.freeDocFrag(outdomdocf);
    DBMS_XSLPROCESSOR.freeProcessor(proc);
END;
/
```

DBMS_XSLPROCESSOR Subprograms

```

PROCESSXSL (p IN Processor, ss IN Stylesheet, xmldoc IN
[DOMDOCUMENT/URL/CLOB])
RETURN DOMDOCUMENTFRAGMENT;

TRANSFORMNODE (n IN DOMNODE, ss IN Stylesheet)
RETURN DOMDocumentFragment;

READ2CLOB (flocation IN VARCHAR2, fname IN VARCHAR2,
csid IN NUMBER:=0)
RETURN CLOB;

FREESTYLESHEET (ss IN Stylesheet);

FREEPROCESSOR (p IN Processor);

```

ORACLE

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

Name	Description
PROCESSXSL()	Transforms the input XML document by using the given stylesheet, and returns the result of the transformation as DOMDocumentFragment. This is an overloaded function. The XML document can be passed to this function by using DOMDocument, a URL, or CLOB.
TRANSFORMNODE()	Transforms a node in a DOM tree by using the given stylesheet, and returns the result of the transformation as DOMDocumentFragment
READ2CLOB()	Reads the content of the fname file (stored in the location flocation) into a CLOB
FREESTYLESHEET()	Releases a STYLESHEET object
FREEPROCESSOR()	Releases a PROCESSOR object

DBMS_XSLPROCESSOR.PROCESSXSL()

```
PROCESSXSL( p IN Processor, ss IN Stylesheet, xmldoc IN DOMDOCUMENT,  
dir IN VARCHAR2, fileName IN VARCHAR2);
```

1

```
PROCESSXSL( p IN Processor, ss IN Stylesheet, url IN VARCHAR2, dir IN  
VARCHAR2, fileName IN VARCHAR2);
```

2

```
PROCESSXSL( p IN Processor, ss IN Stylesheet, xmldf IN  
DOMDOCUMENTFRAGMENT) ;
```

3

```
PROCESSXSL( p IN Processor, ss IN Stylesheet, xmldf IN  
DOMDOCUMENTFRAGMENT, dir IN VARCHAR2, filename IN VARCHAR2);
```

4

```
PROCESSXSL( p IN Processor, ss IN Stylesheet, xmldf IN  
DOMDOCUMENTFRAGMENT, buf IN OUT VARCHAR2);
```

5



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

DBMS_XSLPROCESSOR.PROCESSXSL is an overloaded method. The following is a list of functionalities supported by various versions of **DBMS_XSLPROCESSOR.PROCESSXSL()**:

1. Transforms the input XML document by using the given DomDocument and the stylesheet, and writes the output to the specified file
2. Transforms the input XML document stored in the URL by using the stylesheet and writes the output to the specified file in a specified directory
3. Transforms the given DomDocumentFragment by using the stylesheet and returns the resultant document fragment
4. Transforms the input XML DomDocumentFragment (xmldf) by using the given stylesheet and writes the output to the specified file in the specified directory
5. Transforms the input XML DomDocumentFragment (xmldf) by using the given stylesheet and writes the output to a buffer

Summary

In this lesson, you should have learned to:

- Use DBMS_XMLSTORE to store XML data in relational tables
- Use DBMS_XMLDOM to create a DOM document to access and modify an XML document
- List the W3C recommendations for the DOM
- Describe the working of XML parsers and discuss the difference between DOM and SAX
- Use DBMS_XMLPARSER to access the content and structure of an XML document
- Use DBMS_XSLPROCESSOR to transform an XML document



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

In this lesson, you learned about the various PL/SQL XML APIs and their uses in accessing, manipulating, and transforming XML documents.

DBMS_XMLDOM has the following methods:

- newDOMDocument processes the XMLType instance to create a DOMDocument instance .
- makeNode creates a handle to this DOM document.
- writeToBuffer writes the contents of the node to a buffer.
- getElementsByTagName, item, and getChild access the DOM tree.
- setNodeValue manipulates the contents of a node.

DBMS_XMLPARSER has the following methods:

- newParser constructs a parser instance.
- parse, parseBuffer, parseClob parse the XML document stored in the given URI, buffer, and CLOB, respectively.
- getDocument obtains the DOMDocument interface.

DBMS_XSLPROCESSOR has the following methods:

- `newStyleSheet` obtains a new stylesheet based on an XSL document.
- `newProcessor` creates a new processor.
- `processXSL` applies a stylesheet to an XML document.



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)
- Use XML Stylesheet Language (XSL) transformations
- Review and use XSL Transformations (XSLT)



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

Extensible Markup Language

Extensible Markup Language (XML):

- Describes data objects called XML documents
- Is composed of markup language for structuring data
- Requires custom tags for definition, transmission, validation, and interpretation of data
- Conforms to Standard Generalized Markup Language (SGML)
- Has become a standard way to describe data on the web
- Is processed by XML processors



ORACLE

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

XML is a markup language that provides a universal format for structured documents and data on the web.

Although XML documents look similar to HTML documents, they are very different.

- HTML is a markup language that is primarily used for formatting and displaying text and images in a browser.
- XML is a markup language for structuring data rather than formatting information.

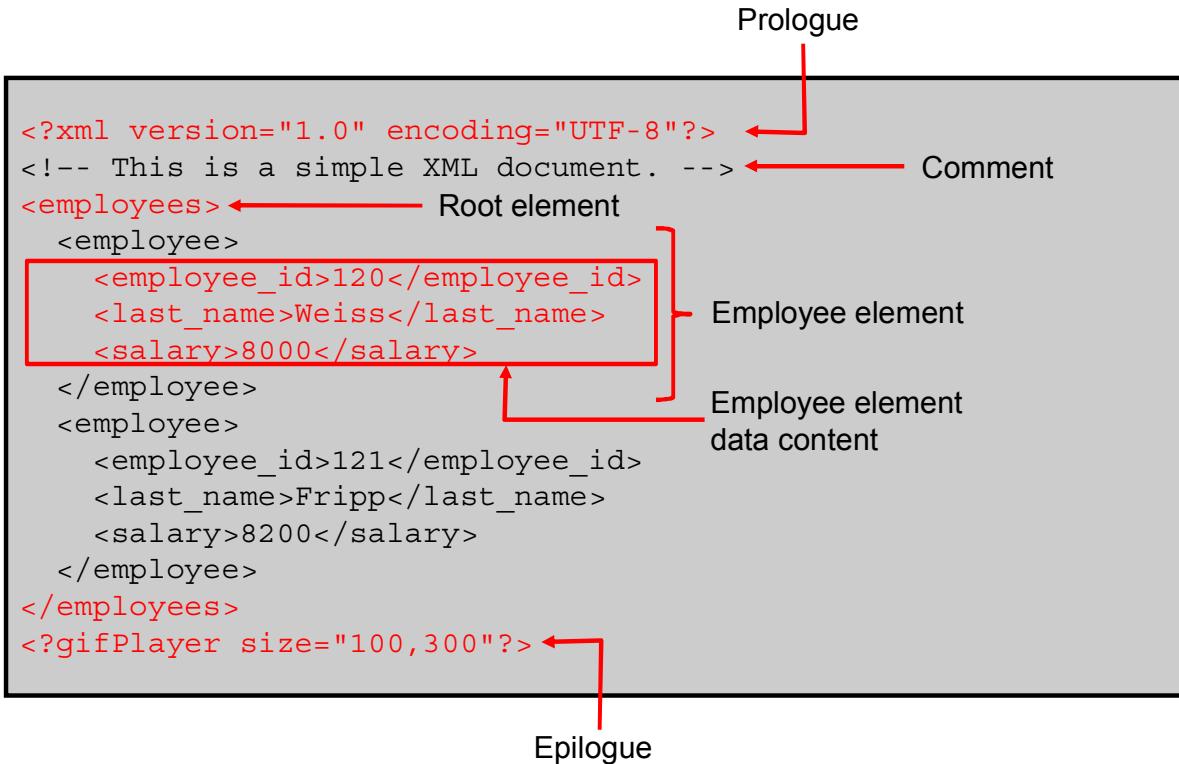
You use XML to create a document that contains structured data that can be used or interpreted by other applications. The format or structure is straightforward and can be used by any person or program that can read text.

Unlike HTML, the tags in XML are extensible, and so you can create your own tags as you need them. HTML has a set of predefined formatting tags that you can use, but you cannot create your own.

XML is part of the World Wide Web Consortium (W3C) standards.

Note: XHTML is a more refined version of HTML. XHTML stands for “Extensible Hypertext Markup Language.”

Example: A Simple XML Document



ORACLE

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

Example: A Simple XML Page

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).

XML Document Structure

An XML document contains the following parts:

- The prologue, which may contain the following information:
 - XML declaration (optional in XML 1.0, mandatory in XML 1.1)
 - Document type definition (DTD), which is required only to validate the document structure
 - Processing instructions and comments, which are optional
- The root element, which is also called the “document element” and contains all other elements
- An epilogue, which contains processing instructions and comments. Processing instructions give commands or information to an application that processes the XML data.

Markup 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.

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. Attributes can be used for the following purposes:

- Describing how the XML document data is encoded or represented
- Indicating where the links or external resources are located

- Identifying and calling external processes such as applets and servlets
- Specifying an element instance in the document for facilitating a rapid search

Note: Attributes always have a value. For example, `name= " "` has an empty string value.

XML Character Data (CDATA)

The CDATA section:

- Is not read by a parser
- Is passed to the application without change
- Is used to contain text that has several XML-restricted characters such as <, >, &, ", and `

```
<script>
if(a &gt; b && b &gt; c)
{print("a is greater than c");}
</script>
```

1

```
<script><! [CDATA[
if(a > b & b > c)
{print("a is greater than c");}
]>
</script>
```

2

ORACLE

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

The text written inside a CDATA section is not read by the parser and is passed to the application without modification.

Characters such as <, >, &, ` , and " are restricted in XML. You must replace them with entities.

If the text inside an element contains many of these restricted characters, replacing them with entities makes the text unreadable and bulky. In such cases, you can write that portion of the text as a CDATA section.

For example, assume that you want an element to contain the following script:

```
if (a > b & b > c)
{
print("a is greater than c")
}
```

Without using CDATA, you must write the element as shown in box 1 in the slide.

Alternatively, you can write the same text as CDATA as shown in box 2.

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 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 DTD to check the document for validity. 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.
- Validations can be done by using a DTD or using 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. What if following only the syntax rules is not quite good enough? Your document must not only follow the XML rules, but 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 considered to be not valid.

You may also need to verify whether these elements have a valid value.

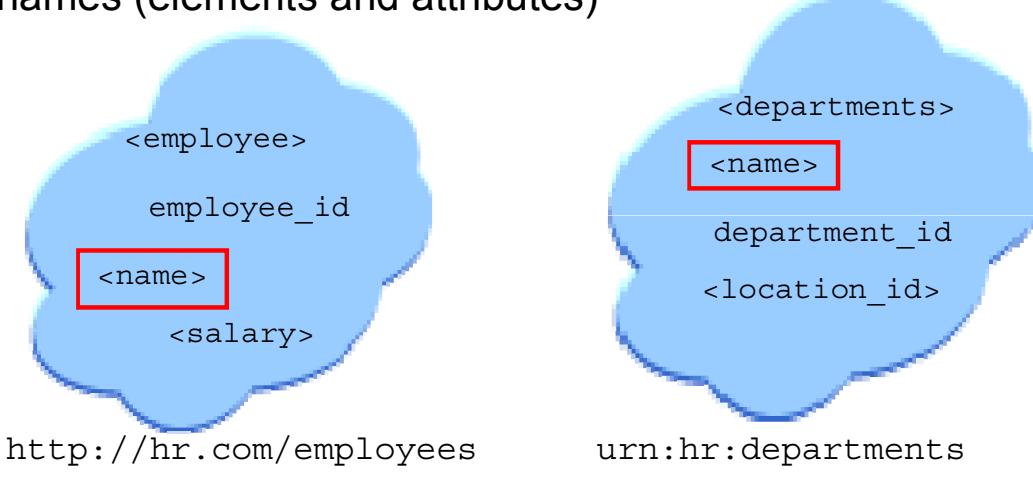
You can perform all these validations by using a DTD. The DTD was the first type of mechanism available for validating XML document structure and content. However, it lacked the capability to perform data type validations on the content. The XML Schema recommendation supports validation for different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

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

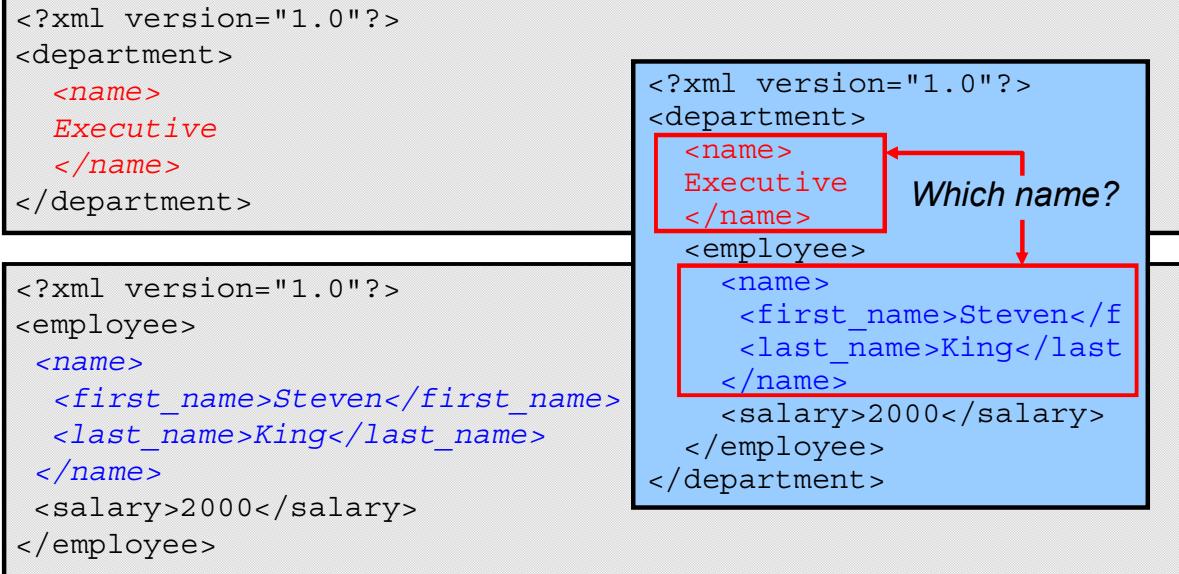
The IRI is a string of characters that can be formatted as:

- A uniform resource indicator (URI) or uniform resource locator (URL) such as the web address `http://hr.com/employees`
Note: The URL or web address represents a unique string and is not checked to be a valid or existing web address.
- A uniform resource name (URN) such as `urn:hr:departments`. A URN starts with the letters `urn`, followed by a colon (:), a Namespace Identifier (NID) (`hr`), a colon (:), and a namespace-specific string (NSS) (`departments`).

The example in the slide shows two collections of XML element and attribute names, each identified by a unique XML Namespaces IRI. 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 collisions or ambiguities 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

Declare an XML Namespace:

- With the `xmlns` attribute in an element start tag
 - Assigned an IRI (URL, URI, or URN) string value
 - Provided with an optional namespace prefix
- With a namespace prefix after `xmlns`: to form qualified element names

```
<dept:department  
    xmlns:dept="urn:hr:department-ns">  
...  
</dept:department>
```

- Without a prefix to form a “default namespace”

```
<department xmlns="http://www.hr.com/departments">  
...  
</department>
```

ORACLE

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

Declaring XML Namespaces in an XML Document

An XML Namespace is declared:

- In the start tag of an XML element by using the `xmlns` attribute, which has:
 - A value representing the unique XML Namespace string (or IRI)
 - An optional prefix, providing an abbreviation for the XML Namespace string
- With a **namespace prefix** to explicitly qualify elements and attribute names. The namespace prefix is declared after `xmlns:` and is used with an XML element to create a **qualified name**. An element name without a prefix is called the **local name**. A prefix can also be used to qualify an XML attribute name.
- Without a prefix to form a **default namespace**, which implicitly qualifies the element name and its children that are not explicitly qualified by a prefix

Note

Attributes that are specified without a prefix are not associated with an XML Namespace.

The examples in the slide show two XML Namespace declarations:

- The `urn:hr:department-ns` string, which is assigned the `dept` prefix. The prefix is used to qualify the `department` element name in the start and end tags.
- The `http://www.hr.com/departments` string, which is a default namespace

An XML Namespace string does not need to reference an actual document or page.

XML Namespace Prefixes

A namespace prefix:

- May contain any XML character, except a colon
- Can be declared multiple times as attributes of a single element, each with different names whose values can be the same or a different string
- Can be overridden in a child element by setting the value to a different string, as in this example:

```
<?xml version="1.0" ?>
<emp:employee xmlns:emp="urn:hr:employee-ns">
    <emp:last_name>King</emp:last_name>
    <emp:address xmlns:emp="urn:hr:address-ns" >
        500 Oracle Parkway
    </emp:address>
</emp:employee>
```



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

An XML Namespace prefix qualifies a local element and attribute name by a particular XML Namespace. The example in the slide qualifies the `last_name` element by using the `emp` prefix representing the `urn:hr:employee-ns` XML Namespace. The `address` element also uses the `emp` prefix, (which is qualified by a different namespace that is `urn:hr:address-ns`), overriding the parent element definition for the XML Namespace. The following example shows two different prefixes (`A` and `B`) being used with the same XML Namespace:

```
<A:department xmlns:A="http://www.a.org/" >
    <A:employees>
        <B:name xmlns:B="http://www.b.org/">
            <B:first_name>Steven</B:first_name>
            <B:last_name>King</B:last_name>
        </B:name>
    </A:employees>
</A:department>
```

Note: It is best to avoid using the same prefix on elements that have a parent-child relationship. It is best to avoid using the same prefix for different namespace declarations because it can be confusing to read and can create problems when processing the document.

XML Namespace Declarations: Example

```
<?xml version="1.0"?>
<department xmlns="urn:hr:department-ns"
             xmlns:emp="urn:hr:employee-ns">
    <name>Executive</name>
    <emp:employee>
        <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>
```



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.

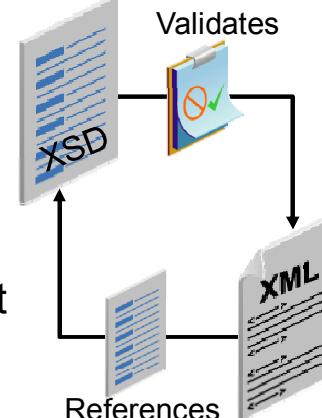
Using XML Namespaces removes the ambiguity for each `<name>` element, which allows them to be processed differently by an XML application. In the example in the slide:

- 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

XML Schema:

- 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 slide shows an XML Schema Document stored separately from the XML instance document that it describes and validates.

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.

Benefits of XML Schemas

XML Schemas:

- Unify both document and data modeling
- Validate XML documents
- Are created using XML
- Support the Namespace Recommendation
- Allow validation of the content of text elements based on built-in or user-defined data types
- Allow modeling of object inheritance and type substitution
- Allow easy creation of complex and reusable content models



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

You can use XML Schema to specify the structure of XML documents and validate them. Although most XML documents are well-formed, they may not be “valid.” A valid document is well-formed and conforms to specific rules defined by either a DTD or an XML Schema. The benefits of using an XML Schema for validation are the following:

- They are easily created by using XML, as defined by the W3C XML Schema language.
- They support the W3C Namespace Recommendation.
- They can be used to validate the content of text elements based on built-in and user-defined data types.
- They allow the creation of complex data type models that can be reused.
- They support object inheritance and substitution in types.

In the web publishing world, a single page that displays the local weather, stock quotes, horoscopes, and specific news channels based on user preferences can involve dozens of queries made to underlying databases and application servers. These queries are made via SQL, the standard object-relational query language, or via some programmatic interface that ultimately calls SQL. Because both SQL and programming languages, such as Java, are strongly typed, they return information that possesses type, structure, constraints, relationships, and so on. The structural and data typing aspects of XML Schema can help exploit the generation of viewable documents from databases.

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

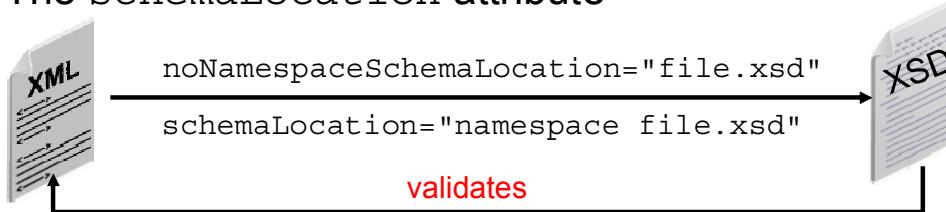
Although the example is a very simple XML Schema document, the resulting XML instance document structure that can be created is not very useful.

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

Validating an XML Document with an XML Schema Document

In the XML instance document, use the XML Namespace <http://www.w3.org/2001/XMLSchema-instance> and reference the XML Schema using either of the following:

- The **noNamespaceSchemaLocation** attribute
- The **schemaLocation** attribute



```
<?xml version="1.0"?>
<departments xmlns:xsi=
  "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="departments.xsd">
  Finance
</departments>
```

XML

ORACLE

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

To validate an XML instance document using an XML Schema, the XML instance document declares the <http://www.w3.org/2001/XMLSchema-instance> XML Namespace in its root element to enable either of the following attributes to be used:

- The **noNamespaceSchemaLocation** attribute, whose value is a file name or URL string identifying the location of the XML Schema document
- The **schemaLocation** attribute, whose value is a namespace/file location pair that identifies the location of an XML Schema using a specific XML Namespace string, a space, and a file location (file name or URL) of the XML Schema document

These attributes are a hint to the XML Parser to validate the XML document using the XML Schema document identified by their value.

Note: The <http://www.w3.org/2001/XMLSchema-instance> namespace must be given a namespace prefix such as `xsi`. The prefix allows the `schemaLocation` or the `noNamespaceSchemaLocation` attribute to be recognized by the XML Parser.

The code example uses the `noNamespaceSchemaLocation` attribute, which means that the XML document does not need to declare any XML Namespace for its elements. For example, the `departments` element need not be qualified by an XML Namespace. However, it is recommended that the `schemaLocation` attribute be used.

Referencing an XML Schema with the schemaLocation Attribute

- The XML Schema defines the targetNamespace value.

```
<?xml version="1.0"?> <!-- departments.xsd -->
<xss:schema
  xmlns:xss="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.hr.com/departments">
  <xss:element name="departments" type="xss:string"/>
</xss:schema>
```

- The XML document references targetNamespace in schemaLocation and the default namespace.

```
<?xml version="1.0"?> <!-- XML document -->
<departments xmlns = "http://www.hr.com/departments"
  xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.hr.com/departments departments.xsd">
  Finance
</departments>
```

XSD

XML

ORACLE

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

It is recommended that you use the schemaLocation attribute rather than the noNamespaceSchemaLocation attribute. Using schemaLocation forces a specified XML Namespace to be used in an XML instance document. This helps to eliminate ambiguity when referencing elements from multiple namespaces in the XML Schema and instance documents.

The XML Schema document in the example, departments . xsd, uses a targetNamespace attribute to define the XML Namespace that must be used in the XML instance document.

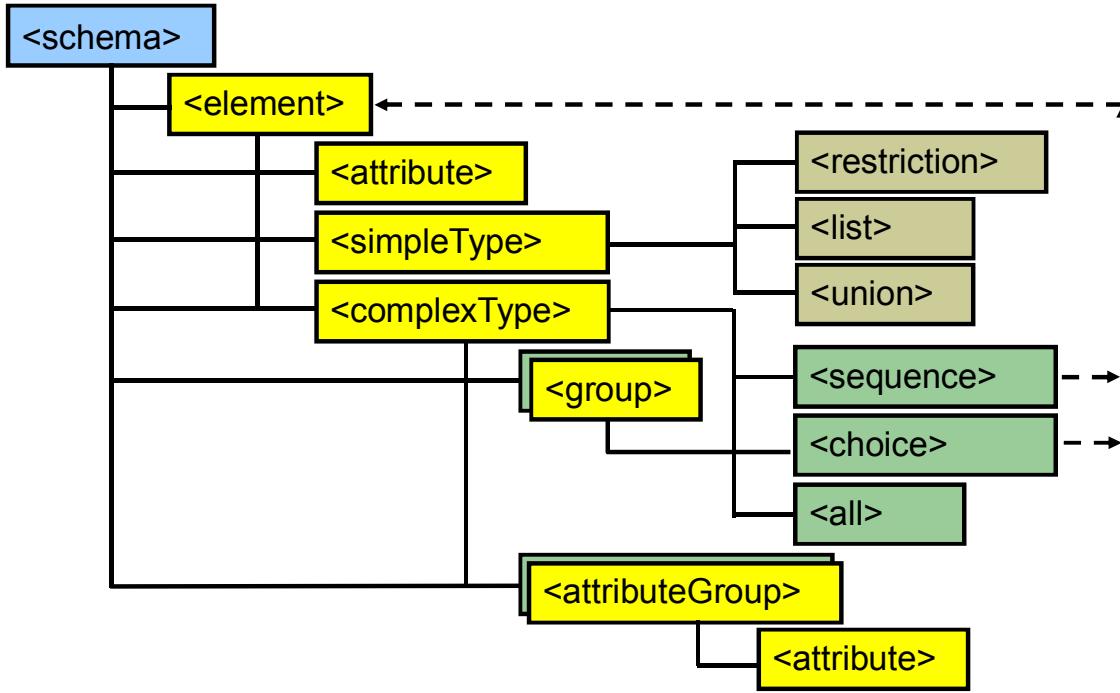
The XML instance document references the XML Schema for validation by using the schemaLocation attribute to specify the following two parts in its value:

- The targetNamespace string as defined in the XML Schema document
- The file name, or URL location, of the XML Schema document

Note: The targetNamespace attribute value is also declared as the default namespace in an xmlns attribute of the <departments> element in the XML instance document.

It is recommended that you declare targetNamespace in an xmlns attribute with a namespace prefix in the XML Schema, thus enabling the XML Schema to unambiguously refer to, reuse, or extend the types and elements declared in the same XML Schema document.

Components of an XML Schema



ORACLE

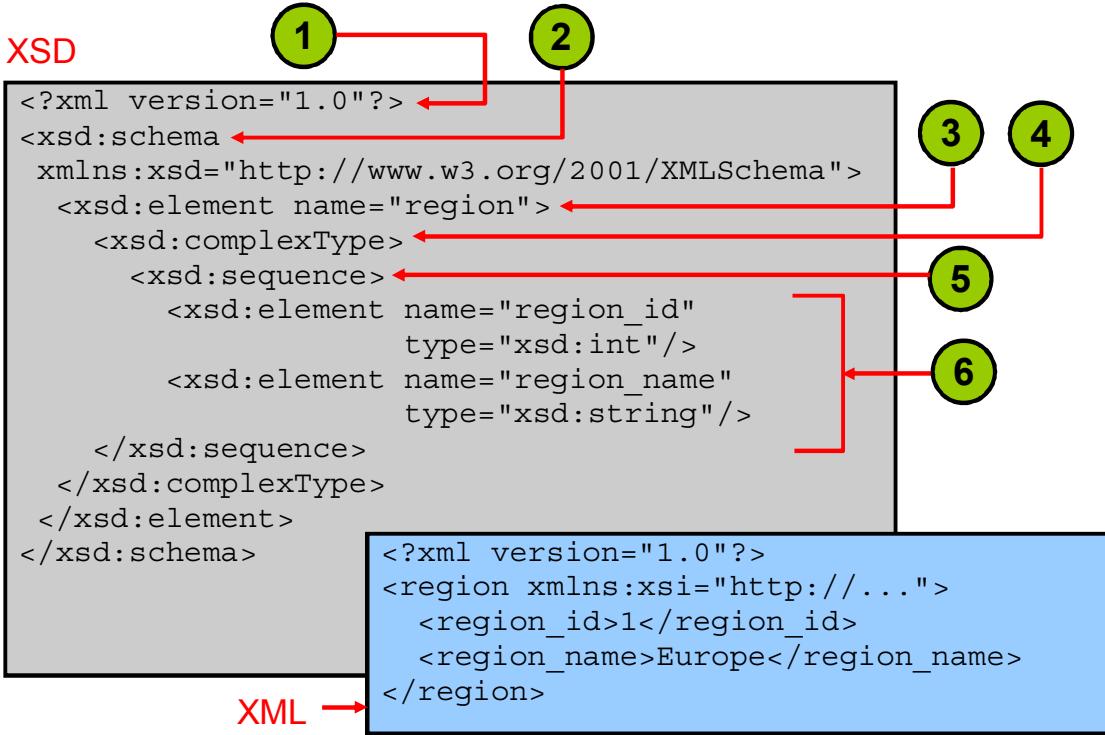
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

<schema> Declaration

Is the root element of every XML Schema. It contains namespace information, defaults, and version.

```
<schema targetNamespace="URI"
        targetNamespace = "URI"
        attributeFormDefault="qualified" | "unqualified"
        elementFormDefault="qualified" | "unqualified"
        version="version number">
    . . .
```

XSD

```
<?xml version="1.0"?
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    . . .
```

XML

Specifies that the *elements* and *data types* used in the schema come from the "<http://www.w3.org/2001/XMLSchema>" namespace and should also be prefixed with xs:

ORACLE

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

The <schema> element is the root element in an XML Schema. It contains namespace information, element or attribute defaults, and any version for the vocabulary. Here is an example:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace = "http://www.oracle.com/namespace"
        xmlns:target="http://www.oracle.com/namespace"
        attributeFormDefault="qualified"
        elementFormDefault="qualified"
        version="1.0">
```

The XML Schema Namespace is <http://www.w3.org/2001/XMLSchema>. The following are valid XML Schema Namespace declarations:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

The namespace prefix has no significance and can be anything that you choose. The prefixes in the preceding examples are most commonly used. An XML Schema declares vocabularies; therefore, the optional `targetNamespace` attribute helps to uniquely identify a vocabulary, and requires a matching namespace declaration to be used with references to declarations within the same XML Schema.

Declaring a Target Namespace

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace = "http://www.oracle.com/namespace"
        xmlns:target="http://www.oracle.com/namespace">
```

Or

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.oracle.com/namespace"
            xmlns:target="http://www.oracle.com/namespace">
```

Note: Declaring the `targetNamespace` value as a default namespace, or using a namespace prefix as `xmlns:target="http://www.oracle.com/namespace"`, in the root element allows the XML Schema document to reference the vocabulary names declared in the same XML Schema document. This is necessary when you declare names that are designed to be referenced by other declarations (and the names are then extended or used to derive types for other components).

Declaring Defaults

You can modify the default values of declarations appearing within the XML Schema by setting the `attributeFormDefault` and `elementFormDefault` attributes to the qualified or unqualified (the default) value. These attributes control the qualification of elements and attributes with a namespace, in the instance document.

An element or attribute is qualified if it has an associated namespace, as in the following example:

```
<emp:employee xmlns:emp="http://www.hr.com/employee">
    <first_name>Steven</first_name>
    <last_name>King</last_name>
</emp:employee>
```

In this example:

- The `<employee>` element is qualified by a namespace
- The `<first_name>` and `<last_name>` elements are unqualified; that is, they have no associated namespace

In the following example, all elements are implicitly qualified by the default namespace:

```
<employee xmlns="http://www.hr.com/employee">
    <first_name>Steven</first_name>
    <last_name>King</last_name>
</employee>
```

The preceding example can be explicitly expressed as shown here:

```
<emp:employee xmlns:emp="http://www.hr.com/employee">
  <emp:first_name>Steven</emp:first_name>
  <emp:last_name>King</emp:last_name>
</emp:employee>
```

Declaring an Element

A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.

- Declare a simple `<element>`:
 - A **name** attribute to specify the tag name
 - A **type** attribute to specify the content allowed

```
<xsd:element name="first_name" type="xsd:string"/>
```

XSD

```
<first_name>Steven</first_name>      <!-- XML -->
```

XML

```
<element  
    name="name-of-element"  
    type="global-type | built-in-type"  
    ref="global-element-name"  
    form="qualified | unqualified"  
    minOccurs="non-negative number"  
    maxOccurs="non-negative number | unbounded"  
    default="default-value"  
    fixed="fixed-value">
```

XSD

ORACLE

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

The XML Schema language uses the `<element>` component to declare an XML element. The attributes of `<element>` are:

- The **name** attribute for specifying the tag name for the element
- The **type** attribute to specify the content type, or the type of data contained in the element
- The **ref** attribute for referencing global type names in the schema document
- The **form** attribute that can be used to override the `elementFormDefault` attribute value used in the `<schema>` element
- The **minOccurs** attribute that specifies the minimum number of times the element must appear in the XML instance document. A value of zero indicates that the element is optional.
- The **maxOccurs** attribute that indicates the maximum number of times the element can appear in the XML instance document. A value of `unbounded` means that there is no limit to the number of element occurrences.
- The **default** attribute that specifies a default value for the element if it does not appear in the XML instance document. The default is not applied to an element with a value.

- The `fixed` attribute, which indicates that the content of the element must be equal to this value

Note: The example specifies that the `first_name` element contains string data. The `string` type is a built-in data type in the XML Schema language.

Built-In XML Schema Data Types

The XML Schema language provides built-in data types such as:

- string type
- int type
- decimal type
- Many others (boolean, float, and so on)

Examples:

```
<xsd:element name="employee_id" type="xsd:int"/> XSD
```

```
<employee_id>100</employee_id> XML
```

```
<xsd:element name="salary" type="xsd:decimal"/> XSD
```

```
<salary>1000.50</salary> XML
```

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

The built-in data types that can be specified in the `type` attribute are listed in the following table:

SimpleType	Example or Explanation
string	"this is a string"
boolean	true, false
float	single-precision floating point
double	double-precision 64-bit floating point
decimal	123.45
integer	-12345; 0; 12345
non-positive-integer	12345; 0
negative-integer	-12345
int	123456789
short	12345

SimpleType	Example or Explanation
byte	123
non-negative-integer	0; 123456
unsigned-long	123456789012345
unsigned-int	1234567801
unsigned-short	12345
unsigned-byte	123
positive-integer	123456
date	2000-02-16
time	15:55:00.000
timeinstant	2000-12-31T05:55:00.000 (5:55 am, Coordinated Universal Time)
timeduration	P5Y1M4D10H20M1.2S (5 years; 1 month; 4 days; 20 minutes; 1.2 seconds)
recurringinstant	-02-16T5:55:00 (February 16th every year at 5:55 AM Coordinated Universal Time)
long	1234567890123
binary	Bit pattern
uri-reference	standard URL
ID*	XML 1.0 Compatibility
IDREF*	XML 1.0 Compatibility
ENTITY*	XML 1.0 Compatibility
NOTATION*	XML 1.0 Compatibility
language	en-US (see xml:lang in XML 1.0)
IDREFS*	XML 1.0 Compatibility
ENTITIES*	XML 1.0 Compatibility
NMOKEN*	XML 1.0 Compatibility
NMOKENS*	XML 1.0 Compatibility

Declaring a <simpleType> Component

<simpleType>:

- Is a derived type that extends built-in or other types
- Provides three primary derived types:
 - <restriction>
 - <list>
 - <union>
- Has facets (properties) such as maxInclusive

```
<xsd:simpleType name="empid">
  <xsd:restriction base="xsd:positiveInteger">
    <!-- for positiveInteger the minimum is 1 -->
    <xsd:maxInclusive value="1000"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="employee_id" type="empid"/>
```

ORACLE

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

The <simpleType> declaration includes atomic or built-in types provided by the XML Schema Recommendation. An example of a built-in type is integer, which requires numeric data in a document instance. A <simpleType> declaration can be:

- For a derived type that extends a built-in or other simpleType declarations
- For an element that contains data, and may not contain attributes or child elements
- Refined by using facets or properties

In the example in the slide:

- A <simpleType> called empid is declared as a <restriction> type
- <restriction> uses the base attribute to identify the built-in type from which <simpleType> is derived
- The base type is positiveInteger, representing nonnegative values
- The maxInclusive facet sets a maximum value of 1000 for this data type
- <element> for employee_id references empid <simpleType> as its data type, restricting the employee_id elements to contain positive integers that are less than or equal to 1000

Note: A valid XML code snippet is:

```
<employee_id>999</employee_id>
```

The following table lists the facets or properties that can be used in `simpleType` declarations:

Facet	Description
enumeration	An allowable value in an enumerated list
fractionDigits	The number of digits to the right of a decimal point in a numeric type
length	The number of items in a list type, or characters in a string type
maxExclusive	A maximum value, excluding the value listed
maxInclusive	A maximum value, including the value listed
maxLength	Maximum number of items in a list type, or characters in a string type
minExclusive	A minimum value, excluding the value listed
minInclusive	A minimum value, including the value listed
minLength	Minimum number of items in a list type, or characters in a string type
pattern	Restriction of string type based on a regular expression
totalDigits	Total number of digits in a numeric value
whiteSpace	Control of how whitespace is treated within a type. Values are preserve, replace, or collapse.

Example: To restrict the `job_id` element value by using the `<enumeration>` facet:

```

<xsd:element name="job_id">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="AC_MGR"/>
      <xsd:enumeration value="AD_PRES"/>
      <xsd:enumeration value="FI_MGR"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

```

The XML element can contain only one of the values specified in the enumeration list, as in this example:

```

...
<job_id>AC_MGR</job_id> <!-- is valid -->
<job_id>XX_MGR</job_id> <!-- is not valid -->
...

```

Declaring <complexType> Components

The <complexType> declaration:

```
<xsd:complexType name="..." mixed="true | false">  
  ...  
</xsd:complexType>
```

- Must be identified by a name attribute if it is global; otherwise, it is an anonymous complex type
- Provides a content model that can contain:
 - Simple content
 - A <sequence> declaration
 - A <choice> declaration
 - A reference to a global <group>
 - An <all> declaration
- Can allow mixed or empty content



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

A <complexType> declaration is a powerful construct that enables you to create simple to complex structures. A <complexType> that is declared globally in the XML Schema document must be identified by a name attribute. A locally declared <complexType> is called an anonymous complex type, where the name attribute is not required. Setting the mixed attribute to true allows both text and element content to be included in the content for the element.

The <complexType> declaration specifies different content models that enable the content to be mixed, empty, or contain element hierarchies. A content model can contain:

- Simple content, for data without child elements
- A <sequence> declaration, to specify an ordered sequence of child elements
- A <choice> declaration, for providing a choice of child elements
- A reference to a global <group>, for reusing a group of elements to define the complex type structure
- An <all> declaration, to allow all elements in the content model to be used in any order

Note: Examples of the <sequence>, <choice>, and empty content models are discussed on subsequent pages.

Declaring a <sequence>

- Defines an ordered sequence of elements
- Must be contained within a <complexType>

```
<xsd:element name="department">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="department_id"
                    type="xsd:int"/>
      <xsd:element name="department_name"
                    type="xsd:string"/>
      <xsd:element name="manager_id"
                    type="xsd:int" minOccurs="0"/>
      <xsd:element name="location_id"
                    type="xsd:int" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```



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

<sequence> is used within a <complexType> to declare an ordered list of elements. The department element in the slide may contain the element sequence of <department_id> followed by <department_name>, optionally followed by <manager_id>, which, in turn, can be optionally followed by <location_id>.

Note: The minOccurs attribute is zero for manager_id and location_id. Either of these elements may be omitted. However, if both appear, they must be entered in the sequence shown in the preceding paragraph. For example, the following XML element is valid:

```
<department>
  <department_id>10</department_id>
  <department_name>Administration</department_name>
</department>
```

However, the following is invalid because <location_id> is before <manager_id>:

```
<department>
  <department_id>20</department_id>
  <department_name>Marketing</department_name>
  <location_id>1800</location_id>  <!-- Error -->
  <manager_id>201</manager_id>
</department>
```

Declaring a <choice>

- Defines a choice of alternative elements
- Must also be contained within a <complexType>

```
<xsd:complexType name="employeeType">
  <xsd:choice>
    <xsd:element name="full_time"
      type="xsd:string"/>
    <xsd:element name="part_time"
      type="xsd:string"/>
  </xsd:choice>
</xsd:complexType>
<xsd:element name="employee">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="full_name"
        type="xsd:string"/>
      <xsd:element name="contract"
        type="employeeType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```



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

<choice> allows one of a selection of components to be included in the XML instance document. <choice> is declared within a <complexType>. Assume that the Human Resources department wants to identify employees as full-time or part-time staff. The example in the slide declares an element for the type of employee, called employeeType, which can contain either a <full_time> or <part_time> element, but not both at the same time. The following example is a snippet of a valid text in an XML document:

```
<employee ...>
  <full_name>Steven King</full_name>
  <contract><full_time>Permanent</full_time></contract>
</employee>
```

The following example is invalid:

```
<employee ...>
  <full_name>Steven King</full_name>
  <contract><!-- Error can't contain both children -->
    <full_time>Permanent</full_time>
    <part_time>Six Months</part_time>
  </contract>
</employee>
```

Declaring an Empty Element

- Can be declared using a `<complexType>` declaration without any elements or a content model
- Typically contains attributes

```
<xsd:element name="departments">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="department"
        maxOccurs="unbounded"/>
      <xsd:complexType>
        <xsd:attribute name="department_id"
          type="xsd:int"/>
        <xsd:attribute name="department_name"
          type="xsd:string"/>
      <xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>
```



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

An empty element does not allow text between the start and end tags. Empty elements are mostly useful for elements with only attributes; that is, they do not have text content.

To declare an empty element without attributes in the XML Schema, use:

```
<xsd:element name="marker">
  <xsd:complexType/>
</xsd:element>
```

The valid XML syntax for `marker` is: `<marker></marker>` or `<marker/>`. The example in the slide declares `departments` containing one or more `department` elements, each with a `department_id` and `department_name` attribute.

Note

Text content is not allowed within each `department` element.

A valid XML instance document, using the XML Schema in the slide, can contain:

```
<departments>
  <department department_id="10"
    department_name="Administration"></department>
  <department department_id="20" department_name="Marketing"/>
</departments>
```

The second `<department>` element does not use the end tag format.

Declaring Attributes

Declare an `<attribute>`:

- Identified by the `name` attribute
- With the `type` attribute restricted to built-in or user-defined simple types

```
<xsd:attribute name="department_id"
                type="xsd:string"/>
```

Full Syntax

```
<attribute
    name="name-of-attribute"
    type="global-type | built-in-type"
    ref="global-attribute-declaration"
    form="qualified | unqualified"
    used="optional | prohibited | required"
    default="default-value"
    fixed="fixed-value">
```



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

Attributes declared by using an `<attribute>` component are identified by a `name` and `type` that is based on a built-in or user-defined `<simpleType>`. An attribute can be declared in three ways:

- As a local type contained within another component such as an `<element>` or `<attributeGroup>`
- As a global type that can be referenced, extended, and reused in a local type or as part of an `<attributeGroup>`
- As a reference to an existing global type

In the slide, `department_id` is a global declaration if it appears as a direct child of the `<schema>` component. Otherwise, it is a local type if nested within an element other than `<schema>`. Other attributes of the `<attribute>` component are:

- `ref`, which derives the type from an existing global attribute declaration
- `form`, which indicates whether the attribute must be qualified by a namespace prefix or not (the default)
- `used`, which indicates that the attribute is `optional` (the default), `required`, or `prohibited` (not allowed)

- `default`, which defines a default value for the attribute when it is optional and not specified
- `fixed`, which defines a constant value for the attribute

Attribute Declarations: Example

1. Based on a <simpleType>:

```
<xsd:attribute name="department_id">
  <xsd:simpleType>
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxInclusive value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

2. With a default value:

```
<xsd:attribute name="department_id"
  type="xsd:int" default="10"/>
```

3. Referencing a global type:

```
<xsd:attribute ref="department_id"/>
```



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

The first example declares the `department_id` attribute using a `<simpleType>` declaration. `<simpleType>` uses a `<restriction>` to constrain the data type values to being positive integer values less than or equal to 100.

The second example declares `department_id` as a simple integer type with a default value of 10. The default value is assumed if the attribute is not included in the XML document.

The third example declares an attribute that references the `department_id` attribute type declared in the first example. The attribute reference example assumes that `department_id` is uniquely declared in the default namespace of the same XML Schema document.

Each of the examples in the slide can be declared globally or nested (locally) as part of:

- An element declaration
- A `<complexType>` declaration. An example of this is shown in the section titled “Declaring an Empty Element,” which contains attribute declarations.
- An `<attributeGroup>` declaration

Validating an XML Document with its XML Schema by Using oraxml

The oraxml Java command-line utility:

- Does not require registering the XML Schema
- Validates an XML instance document against an XML Schema specified in the schemaLocation or noNamespaceSchemaLocation attribute value
- Requires:
 - `xmlparserv2.jar` in CLASSPATH
 - The `-schema` option
 - The XML document file name

```
java oracle.xml.parser.v2.oraxml -schema test.xml
```

The input XML file is parsed without errors using schema validation mode.



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

The oraxml utility, which is a command-line interface for the Oracle XML Parser (in `xmlparserv2.jar`), can be used to validate an XML document against an XML Schema document. Invoke the oraxml utility by using the following command:

```
java oracle.xml.parser.v2.oraxml -schema test.xml
```

Note: The `java` executable located in the Java Runtime Environment (JRE) binary directory must be in PATH. The `xmlparserv2.jar` file located in the JDeveloper library directory must be in CLASSPATH.

The `-schema` option indicates that the `test.xml` file must be validated against the XML Schema specified in the root element of the XML document. For example, if the XML Schema is contained in a file called `test.xsd`, the XML instance document contains the following:

```
<test
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.demo.com/test test.xsd"
  xmlns="http://www.demo.com/test">
  Sample data
</test>
```

XML Path Language

XML Path Language (XPath):

- Is used to find information in an XML document
- 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
- Is designed to be used by XML applications such as XSLT and XPointer



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

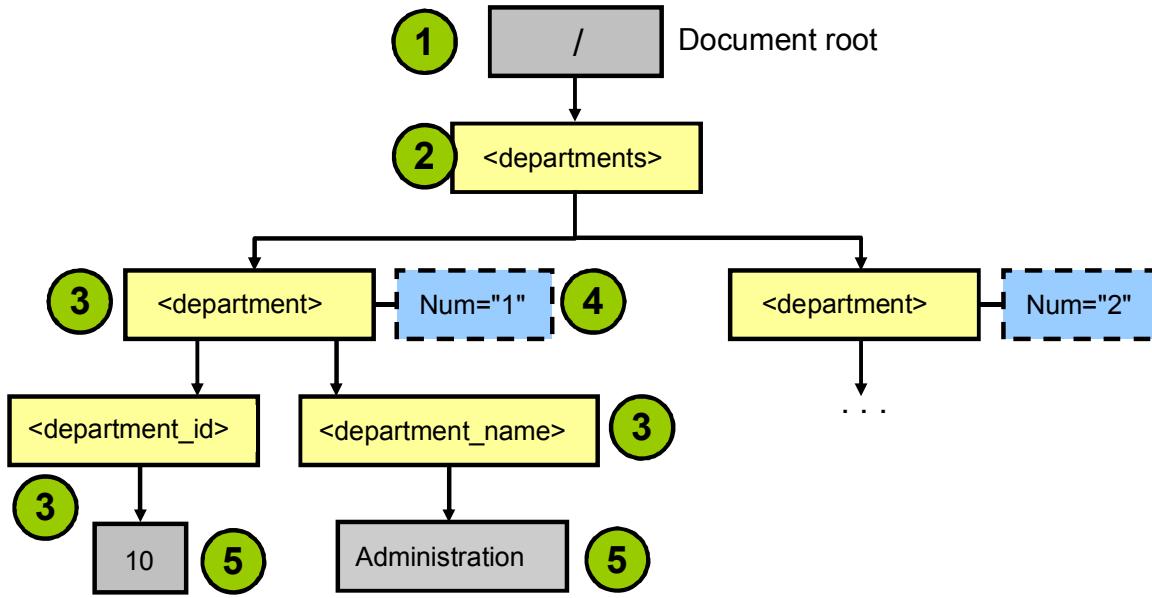
XML Path Language (XPath) is primarily used for addressing parts (nodes) of an XML document. XPath models an XML document as a tree of nodes, and expresses a navigation path through the hierarchical structure of an XML document. XPath:

- 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

In addition to its use for addressing parts of an XML document, XPath fully supports XML Namespaces and provides a natural expression language subset for pattern matching, which is extensively used in the W3C XSLT Recommendation, and XPointer.

Note: XSLT is XML Stylesheet Language Transformation, the language used to transform an XML document into another XML document. XPointer is the XML Pointer Language, which is used as a fragment identifier for any URI reference that locates a resource with a MIME type of `text/xml` or `application/xml`.

XPath Model



ORACLE

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

XPath models an XML document as a tree of nodes. The 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`)

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

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.

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.

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
 - 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 (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

Location Path Expression

- All legal XPath code 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

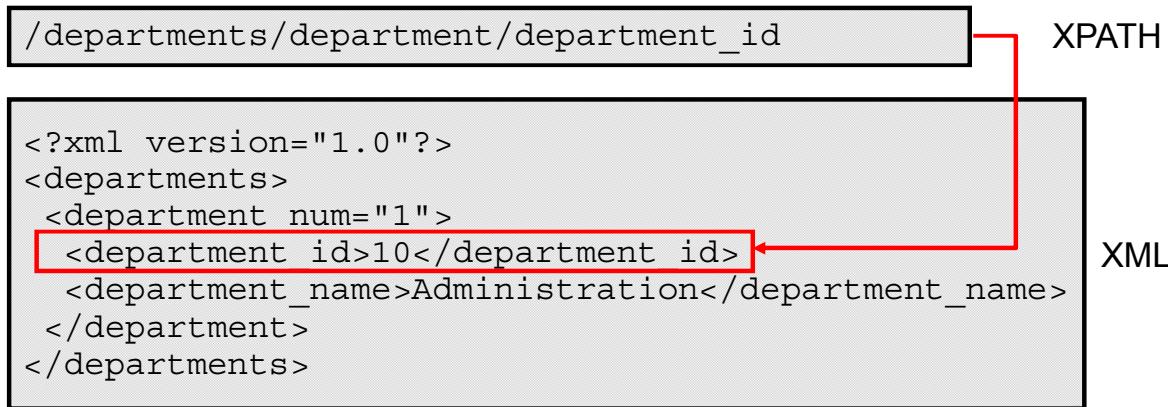


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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**.

Location Path Expression: Example



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>`.

Results of Location Path Expressions

A location path evaluates to a result that is:

- Empty
- A single node
- A set of nodes

Examples:

- The absolute root location path

/

- Child element location steps (examples show location paths relative to the document root)

Relative: departments

Absolute: /departments

Relative: departments/department
Absolute: /departments/department

ORACLE

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

When you evaluate location path expressions, you obtain one of the following results:

- Empty
- A single node
- A set of nodes (more than one node)

The slide shows a skeleton XML document of departments and various location path expressions and their resulting nodes. The relative location paths in the examples assume that the context node is the document root:

- The first example is an absolute location path expression selecting the document root, resulting in the entire document matching the expression.
- The second example shows a relative and an absolute location path selecting the root element node as its result.
- The third example provides a relative and an absolute location path selecting the set of department child elements of the root element.

In the second example, `departments` is called a location step. In the last example, `departments` is the first location step, which is followed by a slash, and then the `department` location step, thus forming a compound XPath expression.

Location Steps in XPath Expressions

Each location step:

- Forms the address of a specific node, or a set of nodes, to be selected
- Comprises three parts:
 - An axis: Defines the tree relationship between the selected nodes and the current node
 - A node test: Identifies a node within an axis
 - Zero or more predicates: Is to further refine the selected node-set

`axis::node-test [predicates]`

- Has both an abbreviated and an unabbreviated form
- Acts like a filter for a selected node or node-set
- Is separated by / to form a compound location path expression



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

As identified, a location path is an important XPath expression that comprises one or more location steps relative to the context node. Each location step is separated by a slash (/) to form a **compound location path** that:

- Specifies the address or location of a node or a set of nodes (node-set)
- Evaluates to a result that is empty, a single node, or a set of nodes

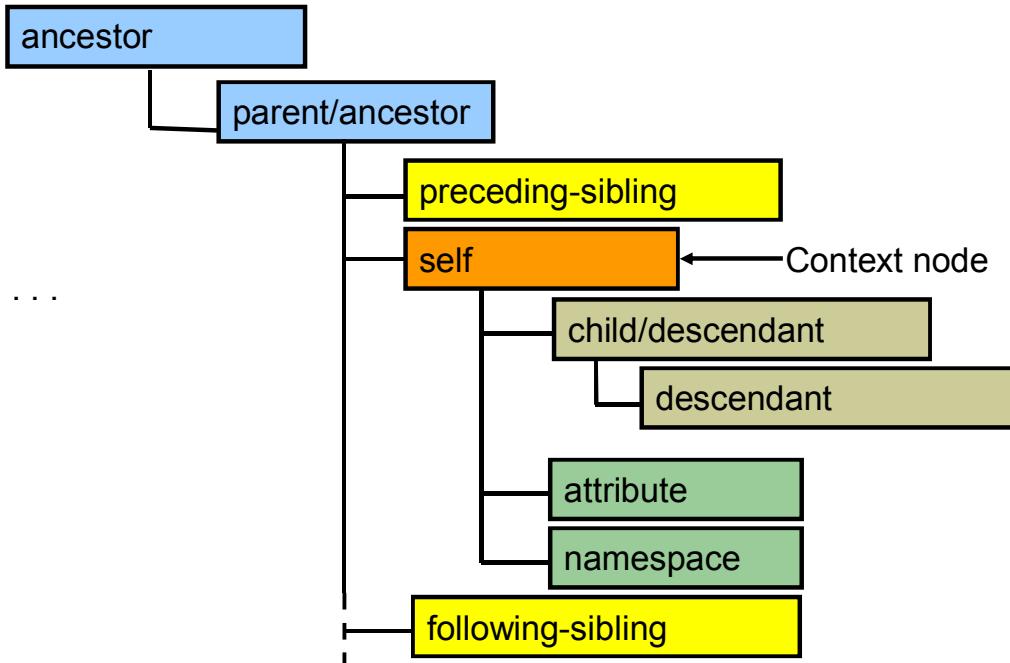
For example, `department/department_name` addresses all the `department_name` child elements of the `department` element that is a child of the context node. Each location step consists of—and is defined by—three parts:

- An **axis**, specifying the relationship between the selected nodes and the context node
- A **node test**, specifying the node type of the selected nodes
- Zero or more **predicates** or conditions that use arbitrary expressions to filter the set of selected nodes

The axis produces a list of nodes that is then filtered using the node test (which is limited to filtering based on node type and node name). The predicates are then applied, if present, to further reduce the node-set.

The location step can be specified in unabbreviated or abbreviated form. The unabbreviated form is seldom used because it is quite verbose, using full axis names followed by a double colon, and then the node test. Axes names are discussed in the section titled “XPath Axes.”

XPath Axes



ORACLE

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

Document tree node axis names (in lowercase) relative to a context node in the diagram:

- child for the children of the context node
- descendant for the descendants of the context node
- parent for the parent of the context node (The document root does not have a parent.)
- ancestor for the ancestors of the context node
- following-sibling for all the following siblings of the context node
- preceding-sibling for all the preceding siblings of the context node
- following for all nodes appearing after the context node in document order
- preceding for all nodes that are before the context node in document order
- attribute for the attributes of the context node (empty for nonelement nodes)
- namespace for the namespace nodes of the context node (empty for nonelement nodes)
- self axis for the context node itself
- descendant-or-self for the context node and its descendants
- ancestor-or-self axis for the context node and its ancestors (and the root node)

Note: The following-sibling and preceding-sibling axes are empty for attribute or namespace context nodes. The descendant, preceding, and following axes exclude attribute or namespace nodes; following excludes descendants and preceding excludes ancestors.

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.

Unabbreviated and Abbreviated Expressions

Abbreviated	Unabbreviated (with axis name)
departments	child::departments
employee/salary	child::employee/child::salary
*	child::*
.	self::node()
..	parent::node()
../salary	parent::node() / child::salary
@id	attribute::id
department/@num	child::department/attribute::num
text()	child::text()
//salary	/descendant-or-self::node() / child::salary



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

Every location path can be expressed by using an abbreviated syntax rather than a verbose unabbreviated syntax. The following unabbreviated XPath expressions are seen in the slide:

- `child::departments` selects the department child elements of the context node.
- `child::employee/child::salary` selects the salary child elements of the employee child elements of the context node.
- `child::*` selects all element children of the context node.
- `self::node()` selects the context node.
- `parent::node()` selects the parent node of the context node.
- `parent::node() / child::salary` selects the salary child element from the parent of the context node.
- `attribute::id` selects the id attribute of the context node.
- `child::department/attribute::num` selects the num attribute from the department child elements of the context node.
- `child::text()` selects all text node children of the context node.
- `/descendant-or-self::node() / child::salary` selects all salary elements of the context node or its descendants.

Note

`child::node()` selects all the children of the context node, whatever their node type. Using `attribute::*` selects all the attributes of the context node.

Using abbreviated syntax:

- `name` selects the `name` element children of the context node.
- `/` selects the document root, which is always the parent of the document element.
- `text()` selects all the text node children of the context node.
- `comment()` selects all the comments node children of the context node.
- `node()` selects all the child node types of the context node.
- `@name` selects the `name` attribute of the context node.
- `@*` selects all the attributes of the context node.
- `.` selects the context node.
- `..` selects the parent of the context node.
- `../@lang` selects the `lang` attribute of the parent of the context node.

The most important abbreviation allows the `child` axis and the double colon (`::`) to be omitted from a location step. In effect, `child` is the default axis.

Other Common XPath Expressions

- `/employee` matches the `employee` element at the document root.
- `department/department_name` matches the `department_name` element as a direct child of the `department` element.
- `departments//department_id` matches the `department_id` element as a descendant of the `departments` element, at any level.
- `.//employee` matches the `employee` element as descendants of the current node.
- `department_name | department_id` matches either the `department_name` or the `department_id` element as children of the context node.

Abbreviated XPath Expression: Examples

```
//last_name
```

Result: <last_name>King</last_name>
<last_name>Kochhar</last_name>

```
employees/employee/name/first_name/text()
```

Result: Steven
Neena

```
employees//salary
```

Result: <salary>24000</salary>
<salary>18000</salary>

```
employees/employee/@employee_id
```

Result: employee_id="100"
employee_id="101"

```
comment()
```

Result: <!-- Employee data -->

Note: The document root is the context node.



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

The examples in the slide are applied to the following XML document:

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

```
</name>
<salary>18000</salary>
</employee>
</employees>
```

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.

XPath Predicates

An XPath predicate:

- 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 using the position() function

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

- May be provided with each location step

```
//department [@num<3] /department_id [.= "10"]
```

- 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 third example shows that each location step can contain a predicate. The example produces the `<department_id>` element containing the text string 10 if it is in a `<department>` element whose `num` attribute value is less than 3.

Note: 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)]
```

Each XPath expression in the slide is applied to the following XML document:

```
<?xml version='1.0' encoding='windows-1252'?>
<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>
```

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<3]/department_id[.= "10"]
<department_id>10</department_id>
- Expression: //department[@num>2 and @num<=4]/department_name
<department_name>Purchasing</department_name>
<department_name>Human Resources</department_name>

XPath Functions

XPath functions:

- Are used in predicates or expressions

```
function-name(arguments, ...)
```

- Have a name followed by parentheses, and zero or more arguments, as in the following example:

```
position()
```

- May return one of the following four types:
 - Boolean
 - Number
 - Node-set
 - String



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

XPath functions are used in predicates and expressions to yield one of the following four result types:

- A **Boolean** function can operate on Boolean expressions to return a `true` or `false` result.
- A **number** function is used to operate on numbers, and returns floating-point values.
- A **node-set** function operates on a set of nodes selected by a location path expression, and returns a result, usually a numeric value.
- A **string** function operates on Unicode string values.

The `position()` function is an example of a node-set function. Functions in each of these categories are discussed on the next few pages.

Note: XPath 2.0 provides more functions than XPath 1.0. XPath 2.0 shares its function library with XQuery 1.0.

Boolean Functions

Function	Description
boolean(o)	Converts to a Boolean
not(b)	Returns true if its argument is false and vice versa
true()	Returns true
false()	Returns false

Example: <department>s without a <manager_id>

```
/departments/department [boolean(not(manager_id))]
```

Partial Results:

```
<department num="12">
  <department_id>120</department_id>
  <department_name>Treasury</department_name>
  <location_id>1700</location_id>
</department>
  ...
  
```



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

Boolean functions manipulate expressions to return either a `true` or `false` value. The `boolean()` function converts its argument to a Boolean value and returns a result, where:

- Nonzero numbers are converted to `true`, and a 0 value returns `false`
- A node-set is `false` if empty; otherwise, it is `true`
- Zero-length strings return `false`; otherwise, a `true` value is returned, as in the following examples:
 - `boolean('false')` returns `true`.
 - `boolean()` returns `false`.

The `not()` function returns the reverse of its Boolean argument. That is, it returns `true` when its argument is `false`, and `false` if its argument is `true`. For example, `not(@salary>10000)` returns `true` if the salary is less than 10000.

The `true()` function always returns `true`.

The `false()` function always returns `false`.

The example in the slide locates the `<department>` elements that do not contain any `<manager_id>` child elements. Although the `boolean()` function is not required in the example, it shows that XPath functions may be nested within the predicate expression.

Note: The example in the slide uses data from the rows of the DEPARTMENTS table in XML format as seen in the partial results.

Node-Set Functions

Function	Description
last()	Returns the context size
position()	Returns the context position
id(o)	Returns elements by their unique ID
count(ns)	Returns the number of nodes in a node-set
local-name(ns)	Returns the local node name (unqualified)

Examples:

```
//department [position()=1]
```

Results: <department num="1">
 <department_id>10</department_id>
 ...
 </department>

```
//department [last()] /department_name
```

Results: <department_name>Payroll</department_name>

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

Node-set functions that manipulate and return information about a set of nodes include:

- The `last()` function that returns the number of nodes in the context node-set
- The `position()` function that returns the context position. For example, `//employee [position()=2]` returns the second employee descendant node.
- The `id()` function that accepts a string containing one or more space-separated ID values as the argument. It returns a node-set containing all the nodes in the document having those ID values, which are declared as an `ID` attribute type in the DTD; that is, the attribute is not named but is declared as a unique `ID` attribute type.
- The `count()` function that is similar to `last()`. It gives a count of the number of nodes in its node-set argument. For example, `count(//employee)` gives you the number of employee elements in the document.
- The `local-name()` function that returns the local name, without a namespace prefix, of the first node in the node-set argument. If no argument is provided, the name of the context node is returned.

The first example in the slide locates the first department element by its position in the node-set. This is equivalent to `//department [1]`. The second example locates the last department in the node-set, which can be written as:

```
//department [count (/department)]/department_name
```

Note: The node-set functions that are not listed in the slide are `namespace-uri()`, which returns the namespace URI string of a node, and `name()`, which returns the qualified name of a node.

String Functions

Function	Description
string(o)	Converts an object to a string
concat(s,s,...)	Concatenates its arguments
substring(s,n,n)	Returns a substring of a string argument from a start position to a length
contains(s,s)	Accepts two arguments and returns true if the first argument contains the second
starts-with(s,s)	Takes two arguments and returns true if the first argument starts with the second; otherwise, returns false
string-length(s)	Returns the length of a string

Example:

```
/departments/department [boolean(not (manager_id))]
```



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

The `string()` function converts any type of data to a string.

The `concat()` function returns the concatenation of its arguments. For example, `concat('This is ', 'cool')` returns the string 'This is cool'.

The `substring()` function takes three arguments: the string to evaluate, a start position, and a length. If the length is omitted, the return string contains all the characters from the start position to the end of the evaluated string, as in the following examples:

- `substring('employee',1,3)` returns the string 'emp'.
- `substring('Administration',6)` returns 'istration'.

The `contains()` function searches for the presence of the second argument within the first. If it is present, it returns `true`; otherwise, it returns `false`:

- `contains('Steve King','ing')` is `true`.
- `contains('Neena Kocchar','charm')` is `false`.

The `starts-with()` function returns `true` if the first argument starts with the second; otherwise, it returns `false`. For example, `starts-with ('Administration', 'Adm')` returns `true`.

The `string-length()` function returns the length of its argument as an integer value. For example, `string-length('King')` returns a value of 4.

The example in the slide returns the `<department>` elements with `<department_name>` child nodes that start with the uppercase letter A.

XSLT and XPath

XSLT:

- Transforms XML into plain text, HTML, or XML
- Specifies transformation rules in elements with attributes that use XPath expressions

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="//department_name">
    <html>
      <body>
        <p><xsl:value-of select=". "/></p>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="*/text()"/>
</xsl:stylesheet>
```



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

An XML Stylesheet Language (XSL) document is an XML document with elements that define transformation rules. The slide shows sample XSL document elements such as:

- The `<xsl:template>` element with a `match` attribute containing an XPath expression that defines the set of nodes to which the template rule applies
- The `<xsl:value-of>` element with the `select` attribute containing an XPath expression
- The `<xsl:value-of>` element, which outputs the text value of the specified node relative to the context node matching its template XPath expression

The `<xsl:template>` element tells the XSLT processor how to locate specific element nodes in the input document, and provides the rules to create the output data as the contents specified between the start and end tags of the `<xsl:template>` element. In the example, you search for `department_name` elements, and use the current node (`.`) to output the value of `department_name`.

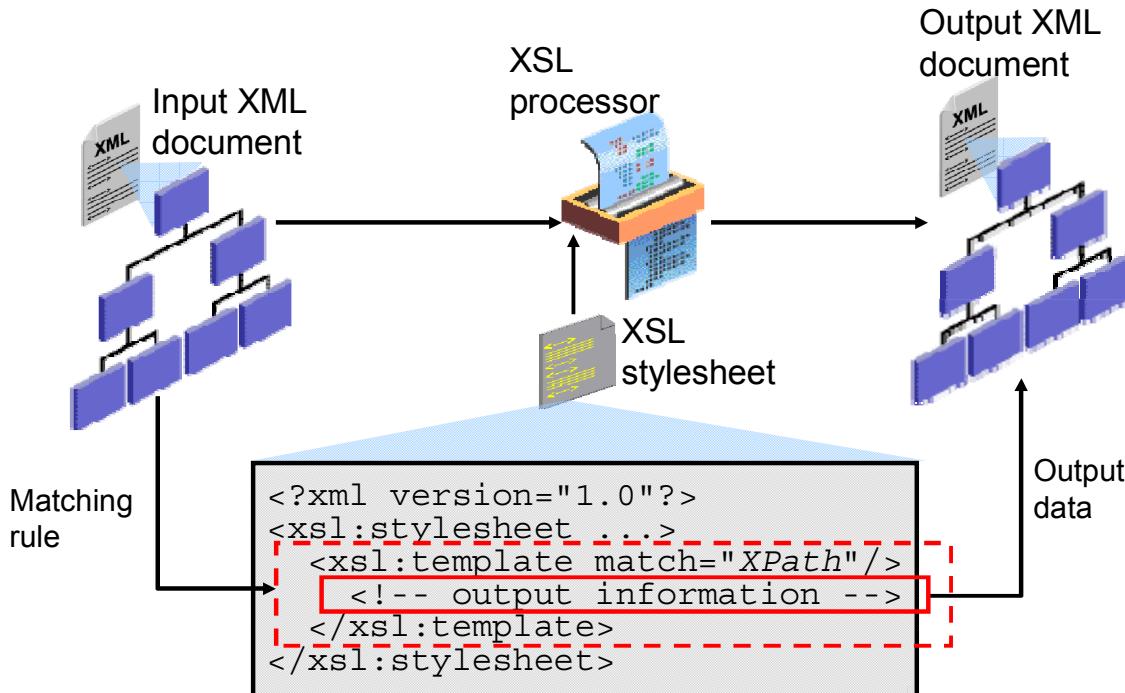
Note

The XSLT Specification states that XSLT processors must provide a default template rule causing text nodes to be output if no matching template rules exist. Suppress the output of unprocessed text nodes by including, at the end of the XSLT Style Sheet, a template rule matching all the text nodes that do not output data. For example, as shown in the slide, use:

```
<xsl:template match="*/text()"/>
```

The example in the slide provides a context where XPath expressions can be used.

XML Stylesheet Language (XSL) Transformations



ORACLE

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

XSL Transformations (XSLT) is an XML application, or XSL Processor, that processes rules specifying how to transform one XML document into another XML document. The rules are contained in an XSLT document called an XSLT Style Sheet, which itself is an XML document and must be well-formed.

The slide shows that an XSLT Style Sheet contains one or more template rules. Each template rule has two parts:

- A match pattern, which is specified as an XPath expression in an attribute of an `<xsl:template>` element
- Template data appearing between the start and end tags of the `<xsl:template>` element that contains the output information. The template data typically contains XML and HTML elements mixed with the XSLT rules.

The XSL Processor compares the elements in the XML input document with the template rule pattern in the XSLT Style Sheet, and writes the template data for matching rules to the output tree, typically, another XML document. Essentially, the transformation process operates on the tree data structure of a source (input) XML document and produces a tree of nodes as its output.

Note

- XSL-FO elements are output elements found in the `<xsl:template>` rules.
- XSLT 2.0 enables you to transform textual data that is not well-formed XML. XSLT 2.0 enables multiple-output documents.

XSLT Style Sheet

An XSLT Style Sheet is an XML document containing:

- An `<xsl:stylesheet>` root element that declares:
 - The `xsl` namespace prefix
 - The `http://www.w3.org/1999/XSL/Transform` mandatory namespace URI
- One or more `<xsl:template>` elements and other XSL elements that define transformation rules

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  ...
  <xsl:template match="/"> ... </xsl:template>
  <xsl:template match="..."> ... </xsl:template>
<xsl:stylesheet>
```



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

An XSLT Style Sheet is an XML document that uses elements from the XSLT vocabulary to describe transformation rules. The document element of every XSLT Style Sheet is the `<xsl:stylesheet>` element, whose content is a set of one or more XSL elements defining the template rules describing the transformation to be performed.

Note

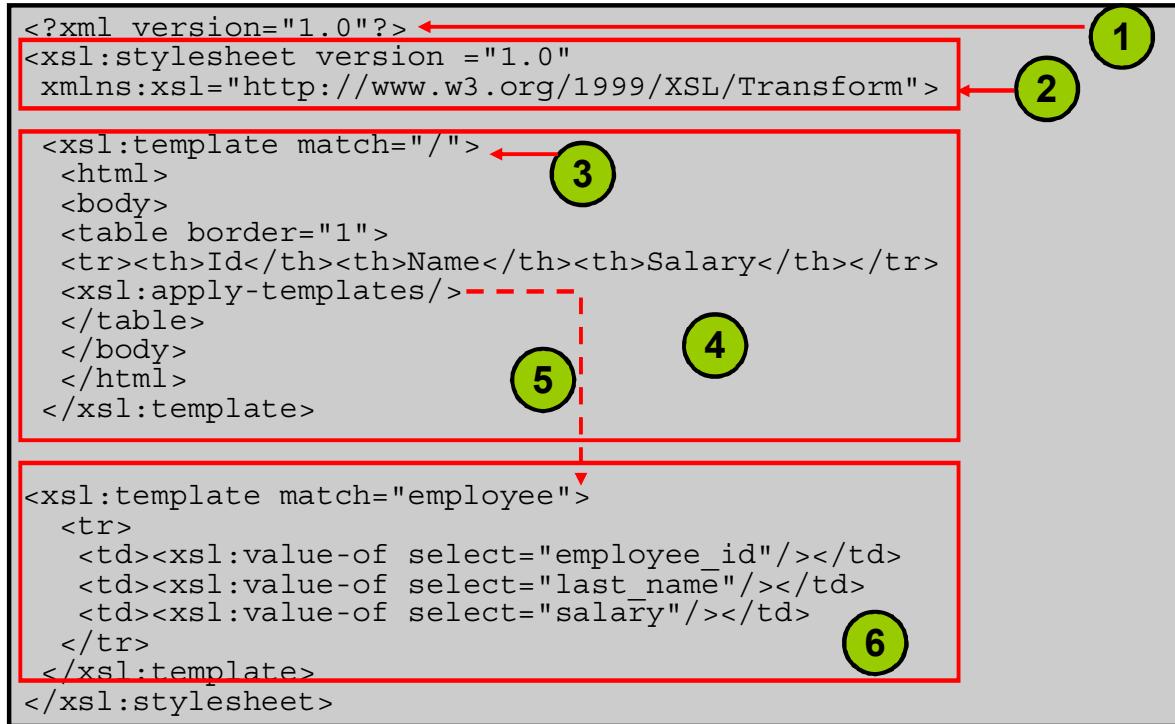
`<xsl:transform>` is a synonym for `<xsl:stylesheet>`.

The `<xsl:stylesheet>` element declares the mandatory namespace attribute `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"` that is used to qualify the elements in an XSLT Style Sheet. If you do not provide this exact namespace prefix and URI, the XSLT Processor simply ignores the rules in `<xsl:template>`, `<xsl:for-each>`, `<xsl:value-of>`, and other XSL elements that are qualified with the `xsl` prefix. Therefore, it does not recognize them as XSLT instructions.

Each stylesheet rule, called a *template*, contains a match pattern specified as an XPath expression that is compared against the nodes in the source XML document. An XSL template rule is represented by an `<xsl:template>` element with a `match` attribute that is assigned a "pattern" string containing an XPath expression.

To use XSLT 2.0, you must specify 2.0 in the `version` attribute of the `<xsl:stylesheet>` element.

XSLT Style Sheet: Example



ORACLE

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

The XSLT document in the example, called a *stylesheet*, shows the following components:

1. The standard XML document declaration, because the stylesheet is an XML document
2. The mandatory XML Namespace declaration using the `xsl` namespace prefix, and the URI `http://www.w3.org/1999/XSL/Transform` namespace
3. The `<xsl:template>` rule matching the root of the source XML document
4. The template data containing the HTML tags to be written to the output document
5. The `<xsl:apply-templates>` element that instructs the XSLT Processor to apply template rules, if any, to the child elements of the root in the source document. The `<xsl:apply-templates>` element is replaced by any output generated for matching child element template rules.
6. The `<xsl:template>` rule for matching the `<employee>` elements in the source document with the template data containing the HTML table row (`<tr>...</tr>`) tags enclosing three table data (`<td>...</td>`) tag pairs. Each table data tag pair encloses an `<xsl:value-of select="..." />` rule that inserts the value of the source document element selected by the XPath expression, which is typically the name of the source document element.

The XML document that uses this XSLT Style Sheet is shown in the slide titled “Viewing the Transformed Document.”

Using an XSLT Style Sheet with an XML Document

An XML document uses an `<?xml-stylesheet...?>` processing instruction:

- After the XML declaration and before the root element of the XML document
- Containing two pseudoattributes:
 - The `type` value is the MIME type for the stylesheet.
 - The `href` value is the URL of the source XSLT Style Sheet document.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<employees>
  ...
</employees>
```



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

An XSLT Style Sheet is applied to an XML document by adding the `xml-stylesheet` processing instruction in the XML document prologue that is before the document root element. The `xml-stylesheet` processing instruction provides two pseudoattributes:

- The `type` value is the Multipurpose Internet Mail Extensions (MIME) media type for the type of stylesheet. For XSL stylesheets, the `type` value must be either `application/xml` or `text/xml`. However, the XSL Processor in Internet Explorer does not recognize `application/xml` as the `type` value, but accepts a value of `text/xsl` as shown in the example in the slide. A MIME media type value of `text/css` is applicable to a Cascading Style Sheet (CSS).
- The `href` value is a URI string referencing the name of the stylesheet document.

The `xml-stylesheet` processing instruction is required for a browser to invoke its XSLT Processor to apply the stylesheet to the XML document tree.

Note: The `type` pseudoattribute value of `text/xsl`, which is a fabrication of Microsoft, is not a formally registered MIME type. However, this course uses the `text/xsl` MIME type to make it easier to observe the results of XSLT. The processing instruction is not required by the Oracle command-line XSLT Processor. Therefore, the MIME type used is irrelevant to the Oracle command-line XSLT Processor.

Viewing the Transformed Document

Perform one of the following:

- Open the XML document in the browser.
- View the `oraxsl` command-line processor output.
- Use JDeveloper.

The diagram illustrates the transformation of an XML document into an XSLT table. On the left, the source XML document is shown:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<employees>
  <employee>
    <employee_id>100</employee_id>
    <last_name>King</last_name>
    <salary>24000</salary>
  </employee>
  <employee>
    <employee_id>101</employee_id>
    <last_name>Kochhar</last_name>
    <salary>18000</salary>
  </employee>
</employees>
```

An arrow points from the XML document to the right, where the resulting XSLT table is displayed in a Mozilla Firefox browser window:

Mozilla Firefox

File Edit View History Bookmarks Tools Help

file:///home/oracle/labs/code_ex/code_07_09_s.xml

Id	Name	Salary
100	King	24000
101	Kochhar	18000

ORACLE

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

The slide shows the source XML document and the result of applying the XSLT Style Sheet from the section titled “XSLT Style Sheet: Example.” When you open the XML document with Microsoft Internet Explorer, it automatically applies the XSLT Style Sheet specified in the XML document processing instruction and renders the result shown.

Alternatively, you can use the `oraxsl` command-line interface to apply an XSLT Style Sheet to an XML document. To invoke `oraxsl`, use:

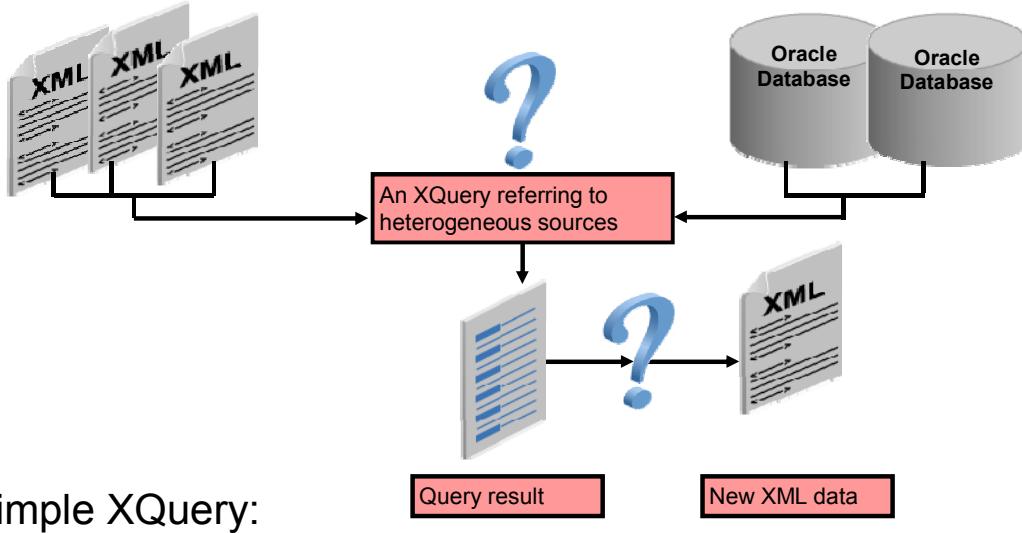
```
set CLASSPATH=D:\JDeveloper\lib\xmlparserv2.jar
java oracle.xml.parser.v2.oraxsl f.xml f.xsl f.html
```

The `oraxsl` parameters in order are:

1. The source XML document (`f.xml`)
2. The XSLT Style Sheet document (`f.xsl`)
3. The output file name (`f.html`). If omitted, the output appears on the standard output.

Note: If the result file contains HTML, you can view it in a browser.

XQuery



A simple XQuery:

```
doc("code_09_dept.xml")/departments/department[department_name  
= 'Purchasing']
```

ORACLE

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

XQuery is an XML query language developed by the World Wide Web Consortium (W3C) and built on other W3C standards such as XML, namespaces, XSLT, XPath, and XML Schema. XQuery 1.0 is a W3C recommendation.

XQuery is designed to query XML data from both XML files and relational databases. Just as SQL is the query language for structured data expressed as relational tables, XQuery is designed to work with the XML data model and be a comprehensive query language for data that is expressed in XML. You can use a single XQuery on multiple and heterogeneous data sources to transform the query result into new XML data.

XQuery consists of a set of possible expressions that are evaluated and return values (which, for XQuery, are sequences).

Example: The slide displays a simple XQuery that selects all the department nodes from the code_09_dept.xml document, where department_name equals "Purchasing." The doc() function is used to specify the XML document that you want to query.

XQuery Terminology and Syntax Rules

- Common terms used in XQuery include:
 - Node, atomic value, item, relationship of nodes, expression, and sequence
- XQuery is case-sensitive.
- Comments are delimited by (: and :).
- Elements, attributes, functions, and variables must be valid XML names.
- The string value can be in single or double quotation marks.
- A variable is defined with \$ followed by a name.



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

XQuery Terminology

XQuery treats XML documents as trees of nodes. The root node is also known as the document node. An atomic value is an instance of one of the built-in data types defined by XML Schema such as integers, strings, dates, and decimals. An atomic value cannot have parent or children nodes. An item may be a node or an atomic value. A relationship in a document between nodes can be seen through the parent, the children, the siblings, the ancestors, and the descendants.

XQuery is a functional language, which is made up of expressions that can be nested and return values. An expression is the basic building block of XQuery.

Sequences are ordered collections of zero or more items. Every value in the XQuery is a sequence. An XQuery sequence can contain items of any XQuery type, which includes numbers, strings, Boolean values, dates, as well as various types of XML node.

XQuery Syntax Rules

The slide shows some of the syntax rules that you must know when writing XQuery code. XQuery keywords and names are case-sensitive, and generally in lowercase. You use scowling and smiley faces to define comments in an XQuery:

(:This is an example. :)

You define an XQuery variable with \$ followed by a name: \$employee

XQuery Expressions

XQuery expressions are of the following types:

- Primary
- Sequence
- FLWOR
- Path
- Arithmetic
- Logical
- Conditional
- Quantified



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

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

- **Primary expression:** Includes literals, variable references, context item expressions, constructors, and function calls
 - A literal is an atomic (scalar) value. XQuery supports two types of literals: numeric and string.
 - A variable name starts with a dollar sign (\$) (for example, \$num).
- **Sequence expression:** Can be constructed using the comma operator; may contain a series of atomic values or nodes
- **FLWOR expression:** Can be used to query XML data and is considered similar to the SELECT - FROM - WHERE statements in SQL
- **Path expression:** Is used to locate nodes in trees using XPath
- **Arithmetic expression:** Is used to write expressions with arithmetic operators
- **Logical expression:** Can be written by using logical operators
- **Conditional expression:** Also supports IF - THEN - ELSE
- **Quantified expression:** Generates a Boolean based on a sequence's content

Primary Expressions: Variables and Literals

- Variables can be used for:
 - Temporary storage of data
 - Manipulation of stored values
 - Reusability
- The variable name starts with a dollar sign:
 - \$num
- A literal is a value that is not represented by an identifier:
 - String literal: \$cname := "SCOTT"
 - Numeric literal: \$ItemCode := 3568
- A constructor creates an XML structure in a query:
 - Direct constructor
 - Computed constructor



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

You can use variables for the following purposes:

- **Temporary storage of data:** Data can be stored temporarily in one or more variables for validating data input and for later processing.
- **Manipulation of stored values:** Variables can be used for calculations and other data manipulations without accessing the database.
- **Reusability:** After they are declared, variables can be used repeatedly in other statements.

A variable name starts with a dollar sign (\$), as in the example: \$num.

A literal is an explicit numeric, character, string, or Boolean value that is not represented by an identifier. There are two types of literals:

- **String literals:** Characters, numerals, spaces, and special symbols that must be enclosed in quotation marks, as in the following examples:
"I", "Hello", "what's up"
- **Numeric literals:** Integers, decimals, and exponents, as in the following examples:
30568, 30568.5, 30568.5E-2

Primary Expressions: Constructors

You can create XML structures in a query by using constructors. These are provided for element, attribute, document, text, comment, and processing instruction nodes. There are two types of constructors:

- **Direct constructor:** The name of the constructed element is a constant. The constructor creates an element node. For example, the XQuery expression `<a>33` constructs the XML element `<a>33`.
- **Computed constructor:** This can also be used to create nodes. A computed constructor begins with a keyword that identifies the type of node to be created: element, attribute, document, text, processing instruction, or comment.
 - A **computed element constructor** creates an element node, allowing both the name and the content of the node to be computed.
 - A **computed attribute constructor** creates a new attribute node with its own node identity.
 - All **document node constructors** are computed constructors. The result of a document node constructor is a new document node with its own node identity.
 - All **text node constructors** are computed constructors. The result of a text node constructor is a new text node with its own node identity. The following example shows the use of computed element and attribute constructors in a simple case in which the names of the constructed nodes are constants.

```
element book {  
attribute isbn { "isbn-0060229357" },  
element title { "Harold and the Purple Crayon" },  
element author {  
    element first { "Crockett" },  
    element last { "Johnson" }  
}  
}
```

Sequence Expressions

- XQuery is a sequence-manipulation language.
- A sequence expression is a list of items that can be XML nodes or simple content (number, date).
- A comma operator is used to concatenate two values or sequences.
- A singleton sequence acts as a single item.
- A sequence expression is never nested.

```
( "Kiran" , "Kumar" , "XML" , "Fundamentals" )
```



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

XQuery is a general sequence-manipulation language. An XQuery sequence expression evaluates to sequences of homogeneous or heterogeneous items. An item can be a simple value (numbers, strings, Boolean values, dates, and so on); it can also be one of the XML node types (such as element, attribute, text, and so on).

The example in the slide shows the construction of a sequence with four atomic values. The comma operator (,) concatenates two values or sequences, and parentheses () facilitate grouping.

A singleton sequence behaves in the same way as a single item:

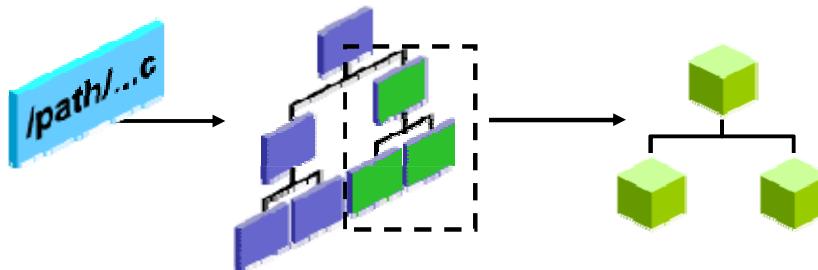
```
(42) = 42
```

You cannot nest sequences. For example, (1, 2, (3, 4, (5), 6), 7) is treated as (1, 2, 3, 4, 5, 6, 7).

Path Expressions

- A path expression is used to locate nodes in trees by using XPath.
- A path always returns a sequence of distinct nodes in document order.
- A path consists of a series of steps.
- A step may contain an axis, a node test, and zero or more predicates.

`axis::node-test [predicates]`



ORACLE

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

Building path expressions is an important part of building XQuery expressions. XQuery path expressions are used to navigate the structure of an XML document and are similar to XPath. The most common task that is performed by using path expressions is to locate nodes by identifying their location in the hierarchy of an XML document.

Each path expression contains a series of one or more steps separated by a slash (/) or double slash (//). Each step returns a sequence of nodes. Each location step consists of and is defined by three types:

- An **axis** specifying the relationship between the selected nodes and the context node. In XQuery, six axes are defined:
 - attribute::
 - child::
 - descendant::
 - descendant or self::
 - parent::
 - self::

- A **node test** specifying the node type of the selected nodes. The node type 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, `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

You can use the vertical bar (`|`) as a separator to combine node tests to create a rule that matches multiple node types. For example, using `/text() | comment()` matches either the text or the comment nodes of the root element.

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

- Zero or more **predicates**, or conditions that use arbitrary expressions to filter the set of selected nodes. A predicate is contained in brackets and 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.
 - 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.

Path: `/departments/department[2]`

Output: `<department num="2">`
 `<department_id>20</department_id>`
 `<department_name>Marketing</department_name>`
`</department>`

- Each location step can contain a predicate. The example produces the `<department_id>` element containing the text string `10` if it is in a `<department>` element whose `num` attribute value is less than `3`.

Path: `//department[@num<3]/department_id[.= "10"]`

Output: `<department_id>10</department_id>`

- You can use the `or` and `and` logical operators to combine predicate expressions.

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

Output: `<department_name>Purchasing</department_name>`
`<department_name>Human Resources</department_name>`

Using Conditional Expressions

```
if test_expr  
then expr1  
else expr2
```

```
for $i in doc("code_09_dept.xml")//department  
where $i/department_id < 30  
return <Result>  
    <department_id> {$i/department_id/text()}  
    </department_id>  
    <department> { if ($i/@num = 1)  
        then "Applicable"  
        else "NA" }  
    </department>  
</Result>
```



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

XQuery supports a conditional expression based on the keywords `if`, `then`, and `else`. Each part of the expression is itself an arbitrary expression. For instance, in the following conditional expression, each of the subexpressions can be any XQuery expression:

```
if ($po1/price < $po2/price)  
then $po1  
else $po2
```

In a conditional expression, `condition_expr` should evaluate to a Boolean value or to a type that can be cast to Boolean. If that Boolean value is `true`, the result of the entire conditional expression is the same as the result of evaluating `expr1`; otherwise, it is the same as the result of evaluating `expr2`. The output for the query in the slide is as follows:

```
<Result>  
    <department_id>10</department_id>  
    <department>Applicable</department>  
</Result>  
<Result>  
    <department_id>20</department_id>  
    <department>NA</department>  
</Result>
```

Summary

In this lesson, you should have learned how to:

- Describe XML, 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)
- Use XML Stylesheet Language (XSL) transformations
- Review and use XSL Transformations (XSLT)



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

Glossary

ORACLE®

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

GLOSSARY

A

Access Control Entities (ACEs) Each access control list (ACL) has a list of access control entities (ACEs). An ACE has the following elements: A Grant that indicates whether or not this ACE grants or denies privileges, a principal (either a user or a role) indicates to whom the ACL applies, and a list of privileges that are granted or denied.

Access Control List (ACL) A security mechanism for the Oracle XML DB resources. It specifies a privilege-based access control for resources to users. Whenever a resource is accessed, a security check is performed. You can have only one ACL per resource.

B

Binary XML storage XMLType data is stored in a post-parse, binary format specifically designed for XML data. 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.

C

Compound documents Are file resources that contain other documents or document fragments. XInclude and XLink are W3C recommendations that provide links to other documents.

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.

D

Data-centric use of XML data, the focus is only on pieces of the total set of XML data within a complete document. You can decide the storage model depending on how data-centric or document-centric your data is (see document-centric).

Database A collection of data, usually in the form of tables or files, under the control of a database management system. See *Database management system*.

Database administrator (DBA) A person within the information technology (or information systems) organization, who is responsible for administering, monitoring, and maintaining the database

Database management system (DBMS) The component of a database that controls all user and system activities related to the core functions of the database such as security checking, tablespace allocation, space management

diffXML parameter document You must create the XML document to be used for the diffXML parameter. The value of the diffXML parameter that is passed to the DBMS_XMLSCHEMA.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.

D

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.

Document links Links provided by the XLink and XInclude standards, which are also supported by Oracle XML DB. XLink and XInclude provide links between documents that may reside inside or outside the repository.

Document Object Model (DOM) The Document Object Model (DOM) recommended by the W3C is a universal API for accessing the structure of XML documents. DOM provides a language-neutral and platform-neutral object model for Web pages and XML documents. DOM describes language-independent and platform-independent interfaces to access and operate on XML components and elements.

Document Type Definition (DTD) provides the definition and relationships of elements contained in an XML document. A DTD validates an XML document.

E

Equipartitioning In 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.

F

File Transfer Protocol (FTP) A service that allows you to transfer files to and from other computers on the Internet. Anyone who has access to the FTP server can transfer the available files to his or her computer.

FLWOR 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.

G

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

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 (see Local XML schema).

H

Hard links A resource cannot be deleted as long as it remains the target of a hard link. If you delete a hard link, the resource is also deleted if it is not versioned and the deleted hard link is the last hard link to the resource. Hard links to ancestor folders are not permitted, as it would introduce cycles. It is this acyclic nature of hard links that gives Oracle XML DB Repository its hierarchical structure.

H

Hybrid storage You can mix storage models by using one model for one part of an XML document and a different model for another part. The mixture of structured and unstructured storage is sometimes called hybrid storage. For example, you can embed CLOB storage within structured storage.

Hypertext Markup Language (HTML) The language used to create HTML pages for the Web using a word processor or text editor

Hypertext Transfer Protocol (HTTP) The first component, the protocol, of a URL address, that is used widely in the Internet and intranet environment. HTTP defines how to interpret information. Other common protocols that you may come across include FTP, news, and gopher. Also See *Uniform Resource Locator*.

I

In-Place XML Schema Evolution 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. You Use the DBMS_XMLSHEMA.INPLACEEVOLVE PL/SQL procedure to perform in-place XML schema evolution (see Copy-Based XML Schema Evolution).

J

J2EE (Java 2 Platform, Enterprise Edition) A Java platform designed for the mainframe-scale computing that is typical of large enterprises. Sun Microsystems (together with industry partners such as IBM) designed J2EE to simplify application development in a thin client-tiered environment. J2EE simplifies application development and decreases the need for programming and programmer training by creating standardized, reusable modular components and by enabling the tier to handle many aspects of programming automatically.

JavaScript JavaScript is the most popular scripting language (A scripting language is a lightweight programming language) on the internet, and works on all major browsers such as Internet Explorer, Firefox, Chrome, Opera, and Safari.

L

Local XML schema The registered XML schema is visible only to the owner (see Global XML schema).

N

Namespace Provides a mechanism to distinguish elements with the same name but different definitions used in the same XML document.

O

Oracle JDeveloper Oracle JDeveloper is a powerful development tool for building, debugging, and deploying J2EE, Web services, and Internet applications.

Oracle Data Pump The import and export utilities provide a simple way for you to transfer data objects between Oracle databases, even if they reside on platforms with different hardware and software configurations.

O

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 are generated from the XML schema and because of the way in which these SQL objects are stored in the database.

Oracle XML Database (XML DB) Oracle XML DB is a high-performance, XML storage and retrieval technology available with the Oracle 11g server. It fully absorbs the W3C XML data model into the Oracle server and provides standard access methods for navigating and querying XML. Oracle XML DB combines the advantages of relational database technology with XML technology.

Oracle XML Repository Enables you to store content in the database in hierarchical structures, as opposed to traditional relational database structures.

Ordered Collection Tables (OCT) 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 contents of such a varray is stored as a set of rows in an ordered collection table (OCT).

P

Parsed character data (PDATA) Textual information comprising markup that describes the data it contains and character data

R

Repository events Are like triggers for Oracle XML DB Repository operations. They enable you to program your application to perform certain actions whenever a particular resource operation is executed.

Repository links A repository folder resource can contain links to other folder or file resources. These links are called repository links. Repository links form the repository's folder-child hierarchical relationships and can be navigated by using the file system–enabled protocols.

S

Simple API for XML (SAX) An event-based API for reading XML documents. A program using SAX reads an XML document as a serialized stream of data, that is, one piece at a time.

Service-Oriented Architecture (SOA) An architecture for the development of loosely coupled distributed applications

*SQL*Loader* A utility provided by Oracle database. It is the primary method for quickly populating Oracle Database tables with data from external files. It has a powerful data parsing engine that puts little limitation on the format of the data in the data file. SQL*Loader can be used to load data across a network, from disk, from tape, or from a named pipe.

Standard Generalized Markup Language (SGML) A markup language for organizing and tagging elements of a document, including headings, paragraphs, tables, and graphics. The elements are marked according to their meaning and relationship to other elements rather than to the format of their presentation.

Service Oriented Application Protocol (SOAP) is an XML-based message format that is an open standard defined by the W3C. It is a mechanism for invoking operations and exchanging documents between applications as an envelope protocol that is embedded in other standard protocols.

S

Simple Object Access Protocol (SOAP) A lightweight, XML-based protocol for exchanging data in a distributed environment. The definition of SOAP is in XML and places no restriction on the format.

Structured Storage The XMLTYPE data is stored relationally as a set of objects. This is also referred to as object-relational storage and object-based persistence. The biggest advantage of structured storage is the performance. This 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.

T

Transforming XML In Oracle XML DB, XMLType instances of the XML data stored in XMLType tables, columns, or views can be transformed (formatted) into HTML, XML, and other mark-up languages by using the XSL stylesheets and the `transform()` XMLType function.

U

Uniform Resource Locator (URL) Text used to identify and address an item in a computer network

Uniform Resource Identifier (URI) A string of characters that identifies an Internet Resource

Universal Description, Discovery, and Integration (UDDI) is an XML-based registry that enables businesses to list themselves and their services on the Internet.

Unstructured Storage The XMLTYPE data is stored in Character Large Object (CLOB) instances. This is also referred to as CLOB storage and text-based persistence. CLOB storage should only be used in limited cases. One use case is when you need "document", which means that you need to maintain the original XML data, byte for byte, in particular, all original white space need to be preserved. Another use case is when you mostly retrieve full document, CLOB storage will provide fast insert and retrieval for full documents.

W

Web service A service that performs discrete business functions. Interaction with a Web service is based on a set of standard protocols and technologies called Web services.

Web Services Description Language (WSDL) uses the XML format for describing Web services. It describes what functionality a Web service offers, how it communicates, and where it is accessible.

World Wide Web Consortium (W3C) The World Wide Web Consortium (W3C) is an international community where Member organizations, a full-time staff, and the public work together to develop Web standards. Led by Web inventor Tim Berners-Lee, W3C's mission is to lead the Web to its full potential. Contact W3C for more information.

X

Extensible Hypertext Markup Language (XHTML) The World Wide Web Consortium (W3C) has defined Extensible Hypertext Markup Language (XHTML) as a successor to HTML. XHTML is designed to conform with XML standards and well-formed document rules, and provide a way to reproduce, subset, and extend HTML documents. An XHTML document is a particular XML document instance that is intended for processing by a Web browser.

XML Extensible Markup Language (XML) describes data objects called XML documents that are composed of markup and data.

XML Developer's Kit (XDK) To help application developers take advantage of XML, Oracle Application Server includes XML Developer's Kit, or XDK. XDK is a standards-based set of components, tools, and utilities that eases the task of building and deploying XML-enabled applications.

XML Entity A unit of data storage that is identified by a case-sensitive name. It represents replacement text to eliminate a lot of typing in XML documents. An entity is referenced by its name prefixed with an ampersand (&) and terminated by a semicolon (;). The ampersand is referred to as an escape character.

XMLIndex A new index 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.

XML language specification Defines the rules that govern XML document structure, and how XML processors must read them.

XML Parser Most browsers have a built-in XML Parser to read and manipulate XML. The parser converts XML into a JavaScript-accessible object (the XML DOM). The XML DOM contains methods (functions) to traverse XML trees, and access, insert, and delete nodes.

XML Schema Provides a way to describe the XML document structure using data type definitions, and uses namespace support.

XML Schema Document (XSD) XML Schema is an XML-based alternative to DTD. An XML schema describes the structure of an XML document. The XML Schema language is also referred to as XML Schema Definition (XSD).

XML/SQL Duality 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.

XML schema evolution Can be used to manage changes to an XML schema. This is useful when the structure or format of XML documents must be changed to reflect the changes in application requirements.

XMLType An abstract native SQL data type for XML data. It 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.

XMLType view Wraps data stored in the relational or object-relational tables in XML formats. Similar to an object view, an XMLType view presents the contents as XMLType instances. The main advantage of using an XMLType view is the ability to experiment with various forms of storage, apart from the object-relational, CLOB, and binary XML storage choices that are available to the XMLType tables.

X

XPath The XML Path Language (XPath) provides syntax for searching an XML document. XPath expressions are used in an XSL stylesheet to match specific nodes.

XQuery An XML query language that provides a data model for XML documents and a set of operators to use with the data model.

Extensible Stylesheet Language (XSL) XSL is implemented by XSL Transformations (XSLT) to specify how to transform an XML document into another document. XSLT uses an XML vocabulary for transforming or formatting XML documents.

XSL Formatting Objects (XSL-FO) An XML application that is used for describing the precise layout of text on a page. XSLT is used to generate XSL-FO elements that are output to another XML document. Additional processing is needed to create the final output form such as a portable document format (PDF) document.

XSL Transformations (XSLT) An XML application that processes rules contained in an XSLT Style Sheet. The rules in an XSLT Style Sheet can be applied to any XML document.

XSLT An XML Stylesheet Language (XSL) document is an XML document with elements that define the transformation rules.